JUNOScript™ API Guide

Release 5.4

Juniper Networks, Inc.
1194 North Mathilda Avenue
Sunnyvale, CA 94089
USA
408-745-2000
www.juniper.net

Part Number: 530-007546-01, Revision 1

JUNOScript API Guide, Release 5.4

Writer: Tony Mauro
Editor: Sonia Saruba
Covers and template design: Edmonds Design

Revision History
8 July 2002—First edition.

Table of Contents

List of Tables

List of Tables

About this Manual

This chapter provides a high-level overview of the *JUNOScript API Guide*:

Objectives

This manual describes how to use the JUNOScript application programming interface (API) to configure or request information from the JUNOScript server running on a Juniper Networks router. The JUNOScript API is a set of Extensible Markup Language (XML) tags that describe hardware and software installed and configured on the router.

This manual documents a specific release of the JUNOScript API, as indicated on the document title page. To obtain additional information about the JUNOScript API—either corrections to or information omitted from this manual—refer to the printed software release notes.

To obtain the most current version of this manual and the most current version of the software release notes, refer to the product documentation page on the Juniper Networks Web site, which is located at http://www.juniper.net/.

To order printed copies of this manual or to order a documentation CD-ROM, which contains this manual, please contact your sales representative.

Audience

This manual is designed for Juniper Networks customers who want to write custom applications for router configuration or monitoring. It assumes that you are familiar with basic terminology and concepts of XML and of XML-parsing utilities such as the Document Object Model (DOM) or Simple API for XML (SAX).

Document Organization

This manual contains the following parts and chapters:

- Preface, “About this Manual” (this chapter), provides a brief description of the contents and organization of this manual and describes how to obtain customer support.
- Part 1, “Overview,” provides an introduction to the JUNOScript API:
  - Chapter 1, “Introduction to the JUNOScript API,” briefly describes the JUNOScript API and XML, and outlines a communications session between the JUNOScript server and a client application.
- Part 2, “Session Control, Operational Requests, and Router Configuration,” describes how to use the JUNOScript API to monitor router status and to read or change router configuration:
  - Chapter 2, “JUNOScript Session Control,” explains how to start, control, and terminate a session with the JUNOScript server running on a Juniper Networks router, and describes the conventions that client applications must obey when exchanging JUNOScript-tagged data with the JUNOScript server.
  - Chapter 3, “Operational Requests,” explains how to use the JUNOScript API to request information about router status—the kind of information provided by operational mode commands in the JUNOS command-line interface (CLI).
  - Chapter 4, “Router Configuration,” describes how to use the JUNOScript API to change router configuration or to request information about the current configuration.
- Part 3, “Write JUNOScript Client Applications,” describes how to write client applications that automate access to the JUNOScript server and parse its output.
  - Chapter 5, “Write a Perl Client Application,” describes how to use the JUNOScript Perl module to speed and simplify implementation of client applications written in Perl. It also describes sample Perl scripts that illustrate how to use the Perl module.
  - Chapter 6, “Write a C Client Application,” illustrates how a C-language client application connects to the JUNOScript server.

This manual also contains an index.

General Document Conventions

This document uses the following text conventions:

- Names of commands, files, and directories are shown in a sans serif font, as are configuration hierarchy levels. The following example refers to the ssh command:

  The client application invokes the ssh command.

- Examples of command output and the contents of files or XML document type definitions (DTDs) are shown in a fixed-width font when it is important to preserve the column alignment, or in sans serif font otherwise. The following example using sans serif font is from the junos-chassis DTD:

  ```
  <!ELEMENT chassis-inventory (chassis?)>
  ```

- Options, which are variable terms for which you substitute appropriate values, are shown in italics. The following example refers to the variable called *identifier*:

  For *identifier*, substitute the name of this instance of the object.

- XML tag names are shown in sans serif font and enclosed in angle brackets, which is the standard XML notation for tags. The angle brackets do not indicate that an element is optional, as they do in the syntax statement for a JUNOS CLI command. The following example refers to tags called <generation> and <local-index>:

  Notice that the JUNOScript server emits the <generation> tag before the <local-index> tag.

- Within a stream of XML tags, XML comments are enclosed within the strings <!-- and -->. The following example includes an XML comment:

  ```
  <configuration>
    <forwarding-options>
      <sampling>
        <disable/>
        <!-- other children of the <sampling> tag -->
      </sampling>
    </forwarding-options>
  </configuration>
  ```

  Outside an XML data set, an ellipsis (…) represents parts of a file or example that are omitted to highlight the remaining elements.

List of Technical Publications

Table 1 lists the software and hardware books for Juniper Networks routers and describes the contents of each book.

Table 1: Juniper Networks Technical Documentation

Book	Description
JUNOS Internet Software Configuration Guides	
Getting Started	Provides an overview of the JUNOS Internet software and describes how to install and upgrade the software. This manual also describes how to configure system management functions and how to configure the chassis, including user accounts, passwords, and redundancy.
Interfaces and Class of Service	Provides an overview of the interface and class-of-service functions of the JUNOS Internet software and describes how to configure the interfaces on the router.
MPLS Applications	Provides an overview of traffic engineering concepts and describes how to configure traffic engineering protocols.
Multicast	Provides an overview of multicast concepts and describes how to configure multicast routing protocols.
Network Management	Provides an overview of network management concepts and describes how to configure various network management features, such as SNMP, accounting options, and cflowd.
Policy Framework	Provides an overview of policy concepts and describes how to configure routing policy, firewall filters, and forwarding options.
Routing and Routing Protocols	Provides an overview of routing concepts and describes how to configure routing, routing instances, and unicast routing protocols.
VPNs	Provides an overview of Layer 2 and Layer 3 Virtual Private Networks (VPNs), describes how to configure VPNs, and provides configuration examples.
JUNOS Internet Software References	
Operational Mode Command Reference	Describes the JUNOS Internet software operational mode commands you use to monitor and troubleshoot Juniper Networks routers.
System Log Messages Reference	Describes how to access and interpret system log messages generated by JUNOS software modules and provides a reference page for each message.
JUNOScript API Documentation	
JUNOScript API Guide	Describes how to use the JUNOScript API to monitor and configure Juniper Networks routers.
JUNOScript API Reference	Provides a reference page for each tag in the JUNOScript API.
JUNOS Internet Software Comprehensive Index	
Comprehensive Index	Provides a complete index of all JUNOS Internet software books and the *JUNOScript API Guide*.
Hardware Documentation	
Hardware Guide	Describes how to install, maintain, and troubleshoot routers and router components. Each router platform (M5 and M10 routers, M20 router, M40 router, M40e router, M160 router, and T640 routing node) has its own hardware guide.
PIC Guide	Describes the router Physical Interface Cards (PICs). Each router platform has its own PIC guide.

Documentation Feedback

We are always interested in hearing from our customers. Please let us know what you like and do not like about the Juniper Networks documentation, and let us know of any suggestions you have for improving the documentation. Also, let us know if you find any mistakes in the documentation. Send your feedback to tech-doc@juniper.net.

How to Request Support

For technical support, contact Juniper Networks at support@juniper.net, or at 1-888-314-JTAC (within the United States) or 408-745-2121 (from outside the United States).

Part 1

Overview

Chapter 1
Introduction to the JUNOScript API

The JUNOScript application programming interface (API) is an Extensible Markup Language (XML) application that Juniper Networks routers use to exchange information with client applications. XML is a metalanguage for defining how to mark the organizational structures and individual items in a data set or document with tags that describe the function of the structures and items. The JUNOScript API defines tags for describing the components and configuration of routers.

Client applications can configure or request information from a router by encoding the request with JUNOScript tags and sending it to the JUNOScript server on the router. (The JUNOScript server is a component of the management daemon [mgd process] running on the router and does not appear as a separate entry in process listings.) The JUNOScript server directs the request to the appropriate software modules within the router, encodes the response in JUNOScript tags or formatted ASCII as requested by the client application, and returns the result to the client application. For example, to request information about the status of a router's interfaces, a client application can send the JUNOScript tag called <get-interface-information>. The JUNOScript server gathers the information and returns it in a tag called <interface-information>.

This manual explains how to use the JUNOScript API to configure Juniper Networks routers or request information about configuration or operation. The main focus is on writing client applications to interact with the JUNOScript server, but you can also use the JUNOScript API to build custom end-user interfaces for configuration and information retrieval and display, such as a Web browser-based interface.

This chapter discusses the following topics:

About XML

XML is a language for defining a set of markers, called tags, that define the function and hierarchical relationships of the parts of a document or data set. The tags look much like Hypertext Markup Language (HTML) tags, but XML is actually a metalanguage used to define tags that best suit the kind of data being marked.

The following sections discuss XML and JUNOScript:

For more details about XML, see *A Technical Introduction to XML* at http://www.xml.com/pub/98/10/guide0.html and the additional reference material at the xml.com site. The official XML specification is available at http://www.w3.org/TR/REC-xml.

XML and JUNOScript Tags

JUNOScript tags obey the XML convention that a tag name indicates the kind of information enclosed by the tag. For example, the name of the JUNOScript <interface-state> tag indicates that the tag contains a description of a router interface's current status, whereas the name of the <input-bytes> tag indicates that its contents specify the number of bytes received.

JUNOScript tag names are enclosed in angle brackets, which is an XML convention. The brackets are a required part of the complete tag name, and are not to be confused with the angle brackets used in the JUNOS Internet software manuals to indicate optional parts of command-line interface (CLI) command strings.

When tagging items in an XML-compliant document or data set, you always enclose the item in paired opening and closing tags. XML is stricter in this respect than HTML, which sometimes uses only opening tags. The following examples show paired opening and closing tags:

```
<interface-state>enabled</interface-state>
<input-bytes>25378</input-bytes>
```

If a tag is *empty*—has no contents—you can represent it either as a pair of opening and closing tags with nothing between them or as a single tag with a forward slash after the tag name. For example, the string <snmp-trap-flag/> represents the same empty tag as <snmp-trap-flag></snmp-trap-flag>.

When discussing tags in text, this manual conventionally uses just the opening tag to represent the complete tagged item. For example, it usually refers to "the <input-bytes> tag" rather than "the <input-bytes></input-bytes> tag element."

Document Type Definition

An XML-tagged document or data set is *structured*, because a set of rules specifies the ordering and interrelationships of the items in it. The rules define the contexts in which each tagged item can—and in some cases must—occur. A file called a *document type definition*, or *DTD*, lists every tag that can appear in the document or data set, defines the parent-child relationships between the tags, and specifies other tag characteristics. The same DTD can apply to many XML documents or data sets.

Advantages of Using the JUNOScript API

The JUNOScript API is a programmatic interface. The JUNOScript DTDs fully document all options for every command and all elements in a configuration statement. JUNOScript tag names clearly indicate the function of an element in a command or configuration statement.

The combination of meaningful tag names and the structural rules in a DTD makes it easy to understand the content and structure of an XML-tagged data set or document. JUNOScript tags make it straightforward for client applications that request information from a router to parse the output and find specific information.

The following example illustrates how the JUNOScript API makes it easier to parse router output and extract the needed information. It compares formatted ASCII and XML-tagged versions of output from a router. The formatted ASCII follows:

```
Physical interface: fxp0, Enabled, Physical link is Up
   Interface index: 4, SNMP ifIndex: 3
```

This is the JUNOScript-tagged version:

```
<interface>
   <name>fxp0</name>
   <admin-status>enabled</admin-status>
   <operational-status>up</operational-status>
   <index>4</index>
   <snmp-index>3</snmp-index>
</interface>
```

When a client application needs to extract a specific piece of data from formatted ASCII output, it must rely on the datum's location, expressed either absolutely or with respect to adjacent strings. Suppose that the client application wants to extract the interface index. It can use a utility such as expect to locate specific strings, but one difficulty is that the number of digits in the interface index is not necessarily predictable. The client application cannot simply read a certain number of characters after the Interface index: label, but must instead extract everything between the label and the subsequent string:

```
, SNMP ifIndex
```

A problem arises if the format or ordering of output changes in a later version of the software, for example, if a Logical index field is added following the interface index number:

```
Physical interface: fxp0, Enabled, Physical link is Up
   Interface index: 4, Logical index: 12, SNMP ifIndex: 3
```

A search for the interface index number that relies on the SNMP ifIndex string now returns an incorrect result. The client application must be updated manually to search for the following string instead:

```
, Logical index
```

In contrast, the structured nature of the JUNOScript-tagged output enables a client application to retrieve the interface index by extracting everything within the opening <index> tag and closing </index> tag. The application does not have to rely on an element's position in the output string, so the JUNOScript server can emit the child tags in any order within the <interface> tags. Adding a new <logical-index> tag in a future release does not affect an application's ability to locate the <index> tag and extract its contents.

Tagged output is also easier to transform into different display formats. For instance, you might want to display different amounts of detail about a given router component at different times. When a router returns formatted ASCII output, you have to design and write special routines and data structures in your display program to extract and store the information needed for a given detail level. In contrast, the inherent structure of JUNOScript output is an ideal basis for a display program's own structures. It is also easy to use the same extraction routine for several levels of detail, simply ignoring the tags you do not need when creating a less detailed display.

Overview of a JUNOScript Session

Communication between the JUNOScript server and a client application is session-based: the two parties explicitly establish a connection before exchanging data and close the connection when they are finished. The streams of JUNOScript tags emitted by the JUNOScript server and a client application each constitute a *well-formed* XML document, because the tag streams obey the structural rules defined in the JUNOScript DTDs for the kind of information they encode. Client applications must produce a well-formed XML document by emitting tags in the required order and only in the legal contexts.

The following list outlines the basic structure of a JUNOScript session. For more specific information, see "Start, Control, and End a JUNOScript Session" on page 14.

1. The client application establishes a connection to the JUNOScript server and opens the JUNOScript session.

2. The JUNOScript server and client application exchange initialization tags, used to determine if they are using compatible versions of the JUNOS Internet software and the JUNOScript API.

3. The client application sends one or more requests to the JUNOScript server and parses its responses.

4. The client application closes the JUNOScript session and the connection to the JUNOScript server.

Part 2

Session Control, Operational Requests, and Router Configuration

Chapter 2

JUNOScript Session Control

This chapter explains how to start and terminate a session with the JUNOScript server, and describes the Extensible Markup Language (XML) tags that client applications and the JUNOScript server use to coordinate information exchange during the session. It discusses the following topics:

General JUNOScript Conventions

A client application must comply with XML and JUNOScript conventions. Compliant applications are easier to maintain if the JUNOS Internet software or the JUNOScript application programming interface (API) changes. The JUNOScript server always obeys the conventions. The following sections describe JUNOScript conventions:

Ordering and Context for Session Control Tags

A *session control* tag is one that delimits the parts of a JUNOScript session. There are tags that indicate the start or end of a session, identify a client request or server response, and signal error conditions. Most session control tags can occur only in certain contexts and in a prescribed order. JUNOScript session controllers include the following:

- <?xml?>—An XML processing instruction (PI), emitted by both the client application and the JUNOScript server as they establish a JUNOScript session. For more information about PIs, see "XML Processing Instructions" on page 12.
- <junoscript>—The root tag for every JUNOScript session, emitted by both the client application and the JUNOScript server as they establish and close the session.
- <rpc>—The container tag that encloses each request emitted by the client application. It can occur only within <junoscript> tags.
- <rpc-reply>—The container tag that encloses each response returned by the JUNOScript server. It can occur only within <junoscript> tags.

For more information about how to use these tags, see "Start, Control, and End a JUNOScript Session" on page 14 and the summary of session control tags in the *JUNOScript API Reference*.

Ordering and Context for Request and Response Tags

A *request* tag is one generated by a client application to request information about a router's current status or configuration or to change the configuration. A request tag corresponds to a JUNOS command-line interface (CLI) operational command or configuration statement. It can occur only within <rpc> session control tags.

A *response* tag represents the JUNOScript server's reply to a request tag and occurs only within <rpc-reply> tags.

The following example represents an exchange in which a client application emits the <get-interface-information> request tag with the <extensive/> flag and the JUNOScript server returns the <interface-information> response tag. (For information about the xmlns attribute, see "JUNOScript Server Response Classes" on page 27.)

Client Application

```
<rpc>
   <get-interface-information>
      <extensive/>
   </get-interface-information>
</rpc>
```

JUNOScript Server

```
<rpc-reply>
   <interface-information xmlns="URL">
      <!- - child tags of the <interface-information> tag - ->
   </interface-information>
</rpc-reply>
```

T1000

A client application can send only one request tag at a time to a particular router, and must not send another request tag until it receives the closing </rpc-reply> tag that represents the end of the JUNOScript server's response to the current request.

Ordering and Context for a Request Tag's Child Tags

Some request tags contain child tags. For configuration request tags, each child tag represents a level in the JUNOS configuration hierarchy. For operational request tags, each child tag represents one of the options you provide on the command line when issuing the equivalent CLI command.

Some request tags require that certain child tags be present. To make a request successfully, a client application must emit the required child tags within the request tag's opening and closing tags. If any of the request tag's children are themselves container tags, the opening tag for each must occur before any of the tags it contains, and the closing tag must occur before the opening tag for another tag element at its hierarchy level.

In most cases, the ordering of child tags at one hierarchy level within a request tag is not significant. The important exception is the *identifier tag* for an element, which distinguishes the element from other elements of its type. It must occur first within the container tag that represents the element. Most often the identifier tag specifies the element name and is called <name>. For more information, see "Tag Mappings for Identifiers" on page 42.

Ordering and Context for a Response Tag's Child Tags

A response tag's child tags represent the individual data items returned by the JUNOScript server for a particular request. At one hierarchy level within a response tag, there is no prescribed order for the child tags, and the set of child tags is subject to change in later releases of the JUNOScript API. Client applications must not rely on the presence or absence of a particular tag in the JUNOScript server's output, nor on the ordering of child tags within a response tag. For the most robust operation, include logic in the client application that handles the absence of expected tags or the presence of unexpected ones as gracefully as possible.

Spaces, Newlines, and Other White Space Characters

The JUNOScript API complies with the XML specification in ignoring spaces, newlines, and other characters that represent white space. Client applications must not depend on leading, trailing, or embedded white space when parsing the tag stream emitted by the JUNOScript server. For more information about white space in XML documents, see the XML specification at http://www.w3.org/TR/REC-xml.

Comments

Comments can appear at any point in the tag stream emitted by the JUNOScript server. Client applications must handle them gracefully but must not depend on their content. Client applications also cannot use comments to convey information to the JUNOScript server, because the server automatically discards any comments it receives.

Because the JUNOScript API is based on XML, comments are enclosed within the strings <!-- and -->, and cannot contain the string -- (two hyphens). For more details about comments, see the XML specification at http://www.w3.org/TR/REC-xml.

The following is an example of an XML comment:

```
<!-- This is a comment. Please ignore it. -->
```

XML Processing Instructions

An XML PI contains information relevant to a particular protocol and has the following form:

```
<?PI-name attributes?>
```

Some PIs emitted during a JUNOScript session include information that a client application needs for correct operation. A prominent example is the <?xml?> tag, which the client application and JUNOScript server each emit at the beginning of every JUNOScript session to specify which version of XML and which character encoding scheme they are using. For more information, see “Emit Initialization PIs and Tags” on page 21.

The JUNOScript server can also emit PIs that the client application does not need to interpret (for example, PIs intended for the JUNOS CLI). If the client application does not understand a PI, it must treat the PI like a comment rather than exiting or generating an error message.

Predefined Entity References

By XML convention, there are two contexts in which certain characters cannot appear in their regular form:

- In the string that appears between the opening and closing parts of a tag element (the value contained by the tag)
- In the string value assigned to an attribute of an opening tag

When including a disallowed character in either context, client applications must substitute the equivalent *predefined entity reference*, which is a string of characters that represents the disallowed character. The JUNOScript server uses the same predefined entity references in its response tags, so the client application must be able to convert them to actual characters when processing response tags.

Table 2 summarizes the mapping between disallowed characters and predefined entity references for strings that appear between the opening and closing tags of a tag element.

Table 2: Predefined Entity Reference Substitutions for Tag Content Values

Disallowed Character	Predefined Entity Reference
&	&amp;
<	&lt;
>	&gt;

Table 3 summarizes the mapping between disallowed characters and predefined entity references for attribute values.

Table 3: Predefined Entity Reference Substitutions for Attribute Values

Disallowed Character	Predefined Entity Reference
&	&amp;
'	&apos;
>	&gt;
<	&lt;
"	&quot;

As an example, suppose that the following string is the value contained by the <condition> tag:

```
if (a<b && b>c) return "Peer's dead"
```

Using the required predefined entity references, the <condition> tag looks like this:

```
<condition>if (a&lt;b &amp;&amp; b&gt;c) return "Peer's dead"</condition>
```

Similarly, if the value for the <example> tag's heading attribute is Peer's "age" <> 40, the opening tag looks like this when the required predefined entity references are used:

```
<example heading="Peer&apos;s &quot;age&quot; &lt;&gt; 40">
```

Start, Control, and End a JUNOScript Session

The JUNOScript server communicates with client applications within the context of a JUNOScript *session*. The server and client explicitly establish a connection and session before exchanging data, and close the session and connection when they are finished. The streams of JUNOScript tags emitted by the JUNOScript server and a client application must each constitute a well-formed XML document by obeying the structural rules defined in the JUNOScript document type definition (DTD) for the kind of information they are exchanging. The client application must emit tags in the required order and only in the allowed contexts.

Client applications access the JUNOScript server using one of the protocols listed in "Supported Access Protocols" on page 14. To authenticate with the JUNOScript server, they use either a JUNOScript-specific mechanism or the protocol's standard authentication mechanism, depending on the protocol. After authentication, the JUNOScript server uses the JUNOS login accounts and classes already configured on the router to determine whether a client application is authorized to make each request.

See the following sections for information about establishing, using, and terminating a connection and JUNOScript session:

For an example of a complete JUNOScript session, see "Example of a JUNOScript Session" on page 31.

Supported Access Protocols

The JUNOScript server accepts connections created using the access protocols listed in Table 4, which also specifies the associated authentication mechanism.

Table 4: Supported Access Protocols and Authentication Mechanisms

Access Protocol	Authentication Mechanism
clear-text, a JUNOScript-specific protocol for sending unencrypted text over a Transmission Control Protocol (TCP) connection	JUNOScript-specific
ssh (secure shell)	Standard ssh
SSL (Secure Sockets Layer)	JUNOScript-specific
telnet	Standard telnet

The SSL and ssh protocols are preferred because they encrypt security information (such as a password) before transmitting it across the network. The clear-text and telnet protocols do not encrypt security information.

For information about each protocol's authentication prerequisites, see "Prerequisites for Establishing a Connection" on page 15. For authentication instructions, see "Connect to the JUNOScript Server" on page 18.

Prerequisites for Establishing a Connection

Both the JUNOScript server and the client application must be able to access the software for the access protocol that the client application uses to create a connection. The JUNOScript server can access the protocols listed in "Supported Access Protocols" on page 14, because the JUNOS Internet software distribution includes them. On most operating systems, client applications can access the software for TCP (used by the JUNOScript-specific clear-text protocol) and the telnet protocol as part of the standard distribution. For information about obtaining ssh software for use by a client application, see http://www.ssh.com and http://www.openssh.com. For information about obtaining SSL software, see http://www.openssl.org.

For information about connection prerequisites, see the following sections:

- Prerequisites for clear-text Connections on page 15
- Prerequisites for ssh Connections on page 16
- Prerequisites for SSL Connections on page 16
- Prerequisites for telnet Connections on page 17

Prerequisites for clear-text Connections

If the client application uses the clear-text protocol to send unencrypted text directly over a TCP connection without using any additional protocol (such as ssh, SSL, or telnet), perform the following procedure to activate the xnm-clear-text service on the JUNOScript server machine. The service listens on port 3221:

1. Enter CLI configuration mode on the JUNOScript server machine and issue the following command:

   ```
   [edit]
   user@host# activate system services xnm-clear-text
   ```

2. Commit the configuration:

   ```
   [edit]
   user@host# commit
   ```

Prerequisites for ssh Connections

The ssh protocol uses public-private key technology. The ssh client software must be installed on the machine where the client application runs. If the ssh private key is encrypted (as is recommended for greater security), the ssh client must be able to access the passphrase used to decrypt the key.

If the client application uses the JUNOScript Perl module described in “Write a Perl Client Application” on page 71, no further action is necessary. As part of the Perl module installation procedure, you install a prerequisites package that includes the necessary ssh software.

If the client application does not use the JUNOScript Perl module, perform the following procedures to enable it to establish ssh connections:

1. Install the ssh client on the machine where the client application runs.

2. If the private key is encrypted (as recommended), use one of the following methods to make the associated passphrase available to the ssh client:

   - Run the ssh-agent program to provide key management.

   - Direct the ssh client to the file on the local disk that stores the passphrase.

   - Include code in the client application that prompts a human user for the passphrase.

Prerequisites for SSL Connections

The SSL protocol uses public-private key technology, which requires a paired private key and authentication certificate. Perform the following procedure to enable a client application to establish SSL connections:

1. Install the SSL client on the machine where the client application runs.

   (Skip this step if the client application uses the JUNOScript Perl module described in “Write a Perl Client Application” on page 71. As part of the Perl module installation procedure, you install a prerequisites package that includes the necessary SSL software.)

2. Obtain an authentication certificate in Privacy Enhanced Mail (PEM) format, in one of two ways:

   - Request a certificate from a Certificate Authority; these agencies usually charge a fee.

   - Issue the following openssl command to generate a self-signed certificate; for information about obtaining the openssl software, see http://www.openssl.org.

     The command writes the certificate and an unencrypted 1024-bit RSA private key to the file called *certificate-file*.pem. The command appears here on two lines only for legibility:

     ```
     % openssl req -x509 -nodes -newkey rsa:1024 -keyout certificate-file.pem \
             -out certificate-file.pem
     ```

3. Enter CLI configuration mode on the JUNOScript server machine and issue the following commands to import the certificate. In the first command, substitute the desired certificate name for the *certificate-name* variable. In the second command, for the *URL-or-path* variable substitute the name of the file that contains the paired certificate and private key, either as a URL or a pathname on the local disk:

   ```
   [edit]
   user@host# edit security certificates local certificate-name

   [edit security certificates local certificate-name]
   user@host# set load-key-file URL-or-path
   ```

   The CLI expects the private key in the specified file (*URL-or-path*) to be unencrypted. If the key is encrypted, the CLI prompts for the passphrase associated with it, decrypts it, and stores the unencrypted version.

4. Issue the following commands to activate the xnm-ssl service, which listens on port 3220. In the last command, substitute the same value for the *certificate-name* variable as in Step 3:

   ```
   [edit security certificates local certificate-name]
   user@host# top

   [edit]
   user@host# edit system services

   [edit system services]
   user@host# activate xnm-ssl

   [edit system services]
   user@host# set xnm-ssl local-certificate certificate-name
   ```

5. Commit the configuration:

   ```
   [edit system services]
   user@host# commit
   ```

Prerequisites for telnet Connections

There are no prerequisites for enabling a client application to establish telnet connections, other than ensuring that both the client application and the JUNOScript server can access the telnet software. For a discussion, see “Prerequisites for Establishing a Connection” on page 15.

Connect to the JUNOScript Server

A client application written in Perl can most efficiently establish a connection and open a JUNOScript session by using the JUNOScript Perl module described in "Write a Perl Client Application" on page 71. For more information, see that chapter.

For a client application that does not use the JUNOScript Perl module, first perform the prerequisite procedures for the access protocol being used, as described in "Prerequisites for Establishing a Connection" on page 15. The supported access protocols are listed in "Supported Access Protocols" on page 14.

The client application must then perform the following steps:

1. Open a socket or other communications channel to the JUNOScript server machine (router) by invoking one of the remote-connection routines appropriate for the combination of programming language and access protocol that the application uses.

2. Provide authentication information as required by the access protocol:

   - If using the clear-text or SSL protocol, authenticate with the JUNOScript server. For instructions, see "Authenticate with the JUNOScript Server" on page 19.

     Note that you must already have activated the authentication software on the JUNOScript server machine as specified in "Prerequisites for clear-text Connections" on page 15 and "Prerequisites for SSL Connections" on page 16.

   - If using the ssh or telnet protocol, use the protocol's built-in authentication mechanism. Then issue the junoscript command to request that the JUNOScript server convert the connection into a JUNOScript session. For a C-code example, see "Write a C Client Application" on page 77.

3. Emit initialization tags as described in "Start the JUNOScript Session" on page 21.

For more information about authentication and establishing a connection, see the following sections:

- Authenticate with the JUNOScript Server on page 19
- Connect to the JUNOScript Server from the CLI on page 20

Authenticate with the JUNOScript Server

A client application that uses the clear-text or SSL protocol must authenticate with the JUNOScript server. The client application begins by emitting the <request-login> tag within <rpc> tags. In the <request-login> tag, it encloses the <username> tag to specify the name of the JUNOS user account under which to establish the connection. The account must already exist on the JUNOScript server machine. The application can choose whether to specify the password for the account as part of the current tag sequence:

- To specify the password along with the JUNOS account name, emit the following tag sequence:

```
<rpc>
   <request-login>
      <username>JUNOS-account</username>
      <challenge-response>password</challenge-response>
   </request-login>
</rpc>
```

  This tag sequence is appropriate if the application automates access to router information and does not interact with human users, or obtains the password from a human user before beginning the authentication process.

- To specify only the JUNOS account name, emit the following tag sequence:

```
<rpc>
   <request-login>
      <username>JUNOS-account</username>
   </request-login>
</rpc>
```

  This tag sequence is appropriate if the application does not obtain the password until the authentication process has already begun. In this case, the JUNOScript server returns the <challenge> tag within <rpc-reply> tags to request the password associated with the account. The tag encloses the string Password: which the client application can forward to the screen as a prompt for a human user. The echo attribute on the <challenge> tag is set to the value no to specify that the password string typed by the user does not echo on the screen. The tag sequence is as follows:

```
<rpc-reply>
   <challenge echo="no">Password:</challenge>
</rpc-reply>
```

  The client application obtains the password and emits the following tag sequence to forward it to the JUNOScript server:

```
<rpc>
   <request-login>
      <username>JUNOS-account</username>
      <challenge-response>password</challenge-response>
   </request-login>
</rpc>
```

After it receives the account name and password, the JUNOScript server emits the <authentication-response> tag to indicate whether the authentication attempt is successful:

- If the password is correct, the authentication attempt succeeds and the JUNOScript server emits the following tag sequence:

```
<rpc-reply>
   <authentication-response>
      <status>success</status>
      <message>JUNOS-account</message>
   </authentication-response>
</rpc-reply>
```

  The *JUNOS-account* is the JUNOS account name under which the connection is established. The JUNOScript session begins as described in "Start the JUNOScript Session" on page 21.

- If the password is not correct or the <request-login> tag stream is otherwise malformed, the authentication attempt fails and the JUNOScript server emits the following tag sequence:

```
<rpc-reply>
   <authentication-response>
      <status>fail</status>
      <message>error-message</message>
   </authentication-response>
</rpc-reply>
```

  The *error-message* is a string explaining why the authentication attempt failed. The JUNOScript server emits the <challenge> tag up to two more times before rejecting the authentication attempt and closing the connection.

Connect to the JUNOScript Server from the CLI

The JUNOScript API is primarily intended for use by client applications, but for testing purposes you can establish an interactive JUNOScript session and type commands in a shell window. To connect to the JUNOScript server from JUNOS CLI operational mode, issue the junoscript command:

```
cli> junoscript
```

To begin a JUNOScript session over the connection, emit the initialization tags described in "Start the JUNOScript Session" on page 21. You can then type sequences of tags that represent operational and configuration operations, as described in "Operational Requests" on page 35 and "Router Configuration" on page 39. To eliminate typing errors, save complete tag sequences in a file and use a cut-and-paste utility to copy the sequences to the shell window.

When you close the connection to the JUNOScript server (for example, by emitting the <request-end-session> and </junoscript> tags), the router completely closes your connection instead of returning you to the CLI operational mode prompt.

Similarly, the JUNOScript server completely closes your connection if there are any typographical or syntax errors in the tag sequence you enter.

Start the JUNOScript Session

Each JUNOScript session begins with a handshake in which the JUNOScript server and the client application specify the versions of XML and the JUNOScript API they are using. Each party parses the version information emitted by the other, using it to determine whether they can communicate successfully. The following sections describe how to start a JUNOScript session:

Emit Initialization PIs and Tags

When the JUNOScript session begins, the client application emits an <?xml?> PI and an opening <junoscript> tag, as described in the following sections.

Emit the <?xml?> PI

The client application begins by emitting an <?xml?> PI with the following syntax:

```
<?xml version="version" encoding="encoding"?>
```

The PI attributes are as follows. For a list of the attribute values that are acceptable in the current version of the JUNOScript API, see “Supported Protocol Versions” on page 25.

- version—The version of XML with which tags emitted by the client application comply
- encoding—The standardized character set that the client application uses and can understand

In the following example of a client application’s <?xml?> PI, the version="1.0" attribute indicates that it is emitting tags that comply with the XML 1.0 specification. The encoding="us-ascii" attribute indicates that the client application is using the 7-bit ASCII character set standardized by the American National Standards Institute (ANSI). For more information about ANSI standards, see http://www.ansi.org.

```
<?xml version="1.0" encoding="us-ascii"?>
```

Emit the Opening <junoscript> Tag

The client application then emits its opening <junoscript> tag, which has the following syntax:

```
<junoscript version="version" hostname="hostname" release="release-code">
```

The tag attributes are as follows. The version attribute is required, but the other attributes are optional. For a list of the attribute values that are acceptable in the current version of the JUNOScript API, see “Supported Protocol Versions” on page 25.

- version—(Required) The version of the JUNOScript API that the client application is using.
- hostname—The name of the machine on which the client application is running. The information is used only when diagnosing problems. The JUNOScript API does not include support for establishing trusted-host relationships or otherwise altering JUNOScript server behavior depending on the client hostname.
- release—The identifier of the JUNOS Internet software release for which the client application is designed. It indicates that the client application can interoperate successfully with a JUNOScript server designed to understand that version of the JUNOS Internet software. In other words, it indicates that the client application emits request tags corresponding to supported features of the indicated JUNOS Internet software version, and knows how to parse response tags that correspond to those features. If you do not include this attribute, the JUNOScript server assumes that the client application can interoperate with its version of the JUNOS Internet software. For more information, see “Verify Compatibility” on page 24.

  For *release-code*, use the standard notation for JUNOS Internet software version numbers. For example, the value 5.3R1 represents the initial version of JUNOS Release 5.3.

In the following example of a client application’s <junoscript> tag, the version="1.0" attribute indicates that it is using JUNOScript version 1.0. The hostname="client1" attribute indicates that the client application is running on the machine called client1. The release="5.3R1" attribute indicates that the router is running the initial version of JUNOS Release 5.3.

```
<junoscript version="1.0" hostname="client1" release="5.3R1">
```

Parse Initialization Tags from the JUNOScript Server

When the JUNOScript session begins, the JUNOScript server emits an <?xml?> PI and an opening <junoscript> tag, as described in the following sections.

Parse the JUNOScript Server’s <?xml?> PI

The syntax for the <?xml?> PI is as follows:

```
<?xml version="version" encoding="encoding"?>
```

The PI attributes are as follows. For a list of the attribute values that are acceptable in the current version of the JUNOScript API, see “Supported Protocol Versions” on page 25.

- version—The version of XML with which tags emitted by the JUNOScript server comply
- encoding—The standardized character set that the JUNOScript server uses and can understand

In the following example of a JUNOScript server's <?xml?> PI, the version="1.0" attribute indicates that the server is emitting tags that comply with the XML 1.0 specification. The encoding="us-ascii" attribute indicates that the server is using the 7-bit ASCII character set standardized by ANSI. For more information about ANSI standards, see http://www.ansi.org.

```
<?xml version="1.0" encoding="us-ascii"?>
```

Parse the JUNOScript Server's Opening <junoscript> Tag

The server then emits its opening <junoscript> tag, which has the following form (the tag appears on multiple lines only for legibility):

```
<junoscript version="version" hostname="hostname" os="JUNOS" release="release-code"
        xmlns="namespace-URL" xmlns:junos="namespace-URL"
        xmlns:xnm="namespace-URL">
```

The tag attributes are as follows.

- version—The version of the JUNOScript API that the JUNOScript server is using.
- hostname—The name of the router on which the JUNOScript server is running.
- os—The operating system of the router on which the JUNOScript server is running. The value is always JUNOS.
- release—The identifier for the version of the JUNOS Internet software from which the JUNOScript server is derived and is designed to understand. It is presumably in use on the router where the JUNOScript server is running. The *release-code* uses the standard notation for JUNOS Internet software version numbers. For example, the value 5.3R1 represents the initial version of JUNOS Release 5.3.
- xmlns—The XML namespace for the tags enclosed by the <junoscript> tag that do not have a prefix on their names (that is, the default namespace for JUNOScript tags). The value is a URL of the form http://xml.juniper.net/xnm/*version*/xnm, where *version* is a string such as 1.1.
- xmlns:junos—The XML namespace for the tags enclosed by the <junoscript> tag that have the junos: prefix on their names. The value is a URL of the form http://xml.juniper.net/junos/*release-code*/junos, where *release-code* is the standard string that represents a release of the JUNOS software, such as 5.3R1 for the initial release of version 5.3.
- xmlns:xnm—The XML namespace for the JUNOScript tags enclosed by the <junoscript> tag that have the xnm: prefix on their names. The value is a URL of the form http://xml.juniper.net/xnm/*version*/xnm, where *version* is a string such as 1.1.

In the following example of a JUNOScript server's opening <junoscript> tag, the version attribute indicates that the server is using JUNOScript version 1.0 and the hostname attribute indicates that the router's name is big-router. The os and release attributes indicate that the router is running the initial version of JUNOS Release 5.3. The xmlns and xmlns:xnm attributes indicate that the default namespace for JUNOScript tags and the namespace for tags that have the xnm: prefix is http://xml.juniper.net/xnm/1.1/xnm. The xmlns:junos attribute indicates that the namespace for tags that have the junos: prefix is http://xml.juniper.net/junos/5.3R1/junos. The tag appears on multiple lines only for legibility.

```
<junoscript version="1.0" hostname="big-router" os="JUNOS" release="5.3R1"
        xmlns="http://xml.juniper.net/xnm/1.1/xnm"
        xmlns:junos="http://xml.juniper.net/junos/5.3R1/junos"
        xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm">
```

Verify Compatibility

Exchanging <?xml?> and <junoscript> tags enables a client application and the JUNOScript server to determine if they are running different versions of a protocol. Different versions are sometimes incompatible, and by JUNOScript convention the party running the later version of a protocol determines how to handle any incompatibility. For fully automated performance, include code in the client application that determines if its version of a protocol is later than that of the JUNOScript server. Decide which of the following options is appropriate when the application's version of a protocol is more recent, and implement the corresponding response:

- Ignore the version difference, and do not alter standard behavior to accommodate the JUNOScript server's version. A version difference does not always imply incompatibility, so this is often a valid response.
- Alter standard behavior to provide backward compatibility to the JUNOScript server. If the client application is running a later version of the JUNOS Internet software, for example, it can choose to emit only tags that represent the software features available in the JUNOScript server's version of the JUNOS Internet software.
- End the JUNOScript session and terminate the connection. This is appropriate if you decide that accommodating the JUNOScript server's version of a protocol is not practical.

Supported Protocol Versions

Table 5 lists the protocol versions supported by version 1.0 of the JUNOScript API and specifies the PI or tag and attribute used to specify the information during JUNOScript session initialization.

Table 5: Supported Protocol Versions

Protocol and Versions	PI or Tag	Attribute
XML 1.0	<?xml?>	version="1.0"
ANSI-standardized 7-bit ASCII character set	<?xml?>	encoding="us-ascii"
JUNOScript 1.0	<junoscript>	version="1.0"
JUNOS Release 5.1 JUNOS Release 5.2 JUNOS Release 5.3 JUNOS Release 5.4	<junoscript>	release="5.1R*x*" release="5.2R*x*" release="5.3R*x*" release="5.4R*x*"

Exchange Tagged Data

The session continues when the client application sends a request to the JUNOScript server. The JUNOScript server does not emit any tags after session initialization, except in response to the client application's requests (or in the rare case that it needs to terminate the JUNOScript session). The following sections describe the exchange of tagged data:

- Send a Request to the JUNOScript Server on page 25
- Parse the JUNOScript Server Response on page 27

Send a Request to the JUNOScript Server

To initiate a request to the JUNOScript server, emit the opening <rpc> tag, followed by the tag or tags that represent a particular request, and the closing </rpc> tag, in that order. Enclose each request in a separate pair of opening <rpc> and closing </rpc> tags. For an example of emitting the <rpc> tag in the context of a complete JUNOScript session, see "Example of a JUNOScript Session" on page 31.

See the following sections for further information:

- JUNOScript Request Classes on page 26
- Include Attributes in the Opening < rpc > Tag on page 27

JUNOScript Request Classes

There are two classes of JUNOScript requests:

- *Operational requests*—Requests for information about router status, which correspond to the JUNOS CLI commands listed in the *JUNOS Internet Software Operational Mode Command Reference*. The JUNOScript API defines a specific request tag for many CLI commands. For example, the <get-interface-information> tag corresponds to the show interfaces command, and the <get-chassis-inventory> tag requests the same information as the show chassis hardware command.

  The following sample request is for detailed information about the interface called ge-2/3/0:

  ```
  <rpc>
    <get-interface-information>
      <interface-name>ge-2/3/0</interface-name>
      <detail/>
    </get-interface-information>
  </rpc>
  ```

  For more information about requesting operational information, see “Operational Requests” on page 35. For a complete list of mappings between tags and CLI commands for the current version of the JUNOScript API, see the *JUNOScript API Reference*.

- *Configuration requests*—Requests to change router configuration or for information about the current configuration, either candidate or committed (the one currently in active use on the router). The candidate and committed configurations diverge when there are uncommitted changes to the candidate configuration.

  Configuration requests correspond to the JUNOS CLI configuration statements described in each of the JUNOS Internet software configuration guides. The JUNOScript API defines a tag for every container and leaf statement in the JUNOS configuration hierarchy.

  The following example requests the information about the [edit system login] level of the current candidate configuration:

  ```
  <rpc>
    <get-configuration>
      <configuration>
        <system>
          <login/>
        </system>
      </configuration>
    </get-configuration>
  </rpc>
  ```

  For more information about router configuration, see “Router Configuration” on page 39. For a summary of the available configuration tags, see the *JUNOScript API Reference*.

Although operational and configuration requests conceptually belong to separate classes, a JUNOScript session does not have distinct modes that correspond to CLI operational and configuration modes. Each request tag is enclosed within its own <rpc> tag, so a client application can freely alternate operational and configuration requests.

The client application can send only one request tag at a time to a particular router, and must not send another request tag until it receives the closing </rpc-reply> tag that represents the end of the JUNOScript server response to the current request.

Include Attributes in the Opening <rpc> Tag

Optionally, a client application can include one or more attributes in the opening <rpc> tag for each request. The client application can freely define attribute names. The JUNOScript server echoes each attribute, unchanged, in the opening <rpc-reply> tag in which it encloses its response. You can use this feature to associate requests and responses by defining an attribute in each opening request tag that assigns a unique identifier. The JUNOScript server echoes the attribute in its opening <rpc-reply> tag, making it easy to map the response to the initiating request.

Parse the JUNOScript Server Response

The JUNOScript server encloses its response to a client request in <rpc-reply> tags. Client applications must include code for parsing the stream of response tags coming from the JUNOScript server, either processing them as they arrive or storing them until the response is complete. See the following sections for further information:

- JUNOScript Server Response Classes on page 27
- Use a Standard API to Parse Response Tags on page 29

JUNOScript Server Response Classes

There are three classes of JUNOScript server responses:

- *Operational responses*—Responses to requests for information about router status. These correspond to the output from JUNOS CLI operational commands as described in the *JUNOS Internet Software Operational Mode Command Reference*. The JUNOScript API defines response tags for all defined JUNOScript operational request tags.

  For example, the JUNOScript server returns the information requested by the <get-interface-information> tag in a response tag called <interface-information>, and the information requested by the <get-chassis-inventory> tag in a response tag called <chassis-inventory>.

  The opening tag for an operational response usually includes the xmlns attribute to define the XML namespace for the enclosed tags that do not have a prefix (such as junos:) before their names. The namespace is a URL of the form http://xml.juniper.net/junos/*release-code*/junos-*category*, where *release-code* is the standard string that represents the release of the JUNOS Internet software that is running on the JUNOScript server machine, and *category* represents the type of information.

The following sample response includes information about the interface called ge-2/3/0. The namespace indicated by the xmlns attribute contains interface information for the initial release of version 5.3 of the JUNOS Internet software. (Note that the opening <interface-information> tag appears on two lines only for legibility.)

```
<rpc-reply>
  <interface-information xmlns="http://xml.juniper.net/junos/5.3R1/junos-interface">
    <physical-interface>
      <name>ge-2/3/0</name>
      <!-- other data tags for the ge-2/3/0 interface -->
    </physical-interface>
  </interface-information>
</rpc-reply>
```

For more information about the contents of operational response tags, see “Operational Requests” on page 35. For a summary of operational response tags, see the *JUNOScript API Reference*.

- *Configuration information responses*—Responses to requests for information about the router’s current configuration. The JUNOScript API defines a tag for every container and leaf statement in the JUNOS configuration hierarchy.

  The following sample response includes the information at the [edit system login] level of the configuration hierarchy. For brevity, the sample shows only one user defined at this level.

  ```
  <rpc-reply>
    <configuration>
      <system>
        <login>
          <user>
            <name>admin</name>
            <full-name>Administrator</full-name>
            <!-- other data tags for the admin user -->
          </user>
        </login>
      </system>
    </configuration>
  </rpc-reply>
  ```

  For more information about router configuration, see “Router Configuration” on page 39. For a summary of the available configuration tags, see the *JUNOScript API Reference*.

- *Configuration change responses*—Responses to requests to change router configuration. In this case, the JUNOScript server indicates success by returning an opening <rpc-reply> and closing </rpc-reply> tag with nothing between them. It does not emit another tag that explicitly signals successful completion of the request.

  For more information about router configuration, see “Router Configuration” on page 39. For a summary of the available configuration tags, see the *JUNOScript API Reference*.

For an example of parsing the <rpc-reply> tag in the context of a complete JUNOScript session, see “Example of a JUNOScript Session” on page 31.

Use a Standard API to Parse Response Tags

Client applications can handle incoming XML tags by feeding them to a parser that implements a standard API such as the Document Object Model (DOM) or Simple API for XML (SAX).

Routines in the DOM accept incoming XML and build a tag hierarchy in the client application's memory. There are also DOM routines for manipulating an existing hierarchy. DOM implementations are available for several programming languages, including C, C++, Perl, and Java. The DOM specification is available at http://www.w3.org/TR/REC-DOM-Level-1. Additional information is available at http://search.cpan.org/doc/TJMATHER/XML-DOM-1.37/lib/XML/DOM.pm (part of the Comprehensive Perl Archive Network [CPAN] Web site).

The version indicator in the URL for the DOM.pm file (1.37 in the preceding URL) is subject to frequent revision. Try substituting higher version numbers if the file cannot be found.

One potential drawback with DOM is that it always builds a hierarchy of tags and data, which can become very large. If a client application needs to handle only a subhierarchy at a time, it can use a parser that implements SAX instead. SAX accepts XML and feeds the tags and data directly to the client application, which must build its own tag hierarchy. For more information about SAX, see http://sax.sourceforge.net.

End the Session and Close the Connection

When a client application is finished making requests, it ends the JUNOScript session by emitting the empty <request-end-session/> tag within <rpc> tags. In response, the JUNOScript server emits the <end-session/> tag enclosed in <rpc-reply> tags. The client application waits to receive this reply before emitting its closing </junoscript> tag. For an example of the exchange of closing tags, see "Example of a JUNOScript Session" on page 31.

The client application can then close the ssh, SSL, or other connection to the JUNOScript server machine. Client applications written in Perl can close the JUNOScript session and connection by using the JUNOScript Perl module described in "Write a Perl Client Application" on page 71. For more information, see that chapter.

Client applications that do not use the JUNOScript Perl module use the routine provided for closing a connection in the standard library for their programming language.

Handle an Error Condition

If the JUNOScript server encounters an error condition that prevents it from processing the current request, it emits an <xnm:error> tag, which encloses child tags that describe the nature of the error. Client applications must be prepared to receive and handle an <xnm:error> tag at any time. The information in any response tags already received that are related to the current request might be incomplete. The client application can include logic for deciding whether to discard or retain the information.

An error can occur while the server is performing any of the following operations, and the server can send a different combination of child tags in each case:

- Processing a request submitted by a client application in a defined request tag
- Processing a command string submitted by a client application in a <command> tag (discussed in "Requests and Responses without Defined JUNOScript Tags" on page 37)
- Opening, locking, committing, or closing a configuration as requested by a client application (discussed in "Router Configuration" on page 39)
- Parsing a configuration file submitted by a client application in a <load-configuration> tag (discussed in "Change the Candidate Configuration" on page 55)

If the JUNOScript server encounters a less serious problem, it can emit an <xnm:warning> tag instead. The usual response for the client application in this case is to log the warning or pass it to the user, but to continue parsing the server's response.

For a description of the child tags that can appear within an <xnm:error> or <xnm:warning> tag to specify the nature of the problem, see the entries for <xnm:error> and <xnm:warning> in the *JUNOScript API Reference.*

Halt a Request

To request that the JUNOScript server stop processing the current request, emit the empty <abort/> tag. The JUNOScript server responds with the empty <abort-acknowledgment/> tag. Depending on the operation being performed, response tags already sent by the JUNOScript server for the halted request are possibly invalid. The client application can include logic for deciding whether to discard or retain them as appropriate.

For more information about the <abort/> and <abort-acknowledgment/> tags, see their entries in the *JUNOScript API Reference.*

Display CLI Output as JUNOScript Tags

To display the output from a JUNOS CLI command as JUNOScript tags rather than the default formatted ASCII, pipe the command to the display xml command. The following example shows the output from the show chassis hardware command issued on an M40 Internet backbone router that is running the initial version of JUNOS Release 5.3:

```
user@host> show chassis hardware | display xml
<rpc-reply>
<chassis-inventory xmlns="http://xml.juniper.net/junos/5.3R1/junos-chassis">
<chassis junos:style="inventory">
<name>Chassis</name>
<serial-number>00118</serial-number>
<description>M40</description>
<chassis-module>
<name>Backplane</name>
<version>REV 06</version>
<part-number>710-000073</part-number>
<serial-number>AA2049</serial-number>
</chassis-module>
<chassis-module>
<name>Power Supply A</name>
...
</chassis-module>
...
</chassis>
</chassis-inventory>
</rpc-reply>
```

Example of a JUNOScript Session

This section describes the sequence of tags in a sample JUNOScript session. The client application begins by establishing a connection to a JUNOScript server.

The two parties then exchange initialization PIs and tags, as shown in the following example. Note that the JUNOScript server's <junoscript> tag appears on multiple lines for legibility only. The server does not actually insert a newline character into the list of attributes. For a detailed discussion of the <?xml?> PI and <junoscript> tag, see "Start the JUNOScript Session" on page 21.

Client Application

```
<?xml version="1.0" encoding="us-ascii"?>
<junoscript version="1.0" release="5.3R1">
```

JUNOScript Server

```
<?xml version="1.0" encoding="us-ascii"?>
<junoscript version="1.0" hostname="router1"
        os="JUNOS" release="5.3R1"
        xmlns="URL" xmlns:junos="URL"
        xmlns:xnm="URL">
```

T10011

The client application now emits the <get-chassis-inventory> tag to request information about the router's chassis hardware. The JUNOScript server returns the requested information in the <chassis-inventory> tag. In the following example, tags appear indented and on separate lines for legibility only. Client applications do not need to include newlines, spaces, or other white space characters in the tag stream they send to the JUNOScript server, because the server automatically discards such characters. Also, client applications can issue all tags that constitute a request within a single routine such as the C-language write() routine, or can invoke a separate routine for each tag or group of tags.

Client Application

```
<rpc>
   <get-chassis-inventory>
      <detail/>
   </get-chassis-inventory>
</rpc>
```

JUNOScript Server

```
<rpc-reply>
   <chassis-inventory xmlns="URL">
      <chassis>
         <name>Chassis</name>
         <serial-number>1122</serial-number>
         <description>M10</description>
         <chassis-module>
            <name>Midplane</name>
            <!- - other child tags for the Midplane chassis module - ->
         </chassis-module>
         <!- - tags for other chassis modules - ->
      </chassis>
   </chassis-inventory>
</rpc-reply>
```

T1002

The client application then prepares to create a new privilege class called network-mgmt at the [edit system login class] level of the configuration hierarchy. It begins by using the <lock-configuration/> tag to prevent any other users or applications from altering the candidate configuration at the same time. To confirm that the candidate configuration is locked, the JUNOScript server returns an <rpc-reply> and an </rpc-reply> tag with nothing between them.

Client Application

```
<rpc>
   <lock-configuration/>
</rpc>
```

JUNOScript Server

```
<rpc-reply></rpc-reply>
```

T1003

The client application emits the tags that define the new network-mgmt privilege class, commits them, and unlocks (and by implication closes) the configuration. As when it opens the configuration, the JUNOScript server confirms successful receipt, commitment, and closure of the configuration only by returning an opening <rpc-reply> and closing </rpc-reply> tag with nothing between them, not with a more explicit signal. (You do not need to understand the meaning of all tags at this point. For more information about configuring a router, see "Router Configuration" on page 39.)

Client Application	JUNOScript Server
<rpc>	
<load-configuration>	
<configuration>	
<system>	
<login>	
<class>	
<name>network-mgmt</name>	
<permissions>configure</permissions>	
<permissions>snmp</permissions>	
<permissions>system</permissions>	
</class>	
</login>	
</system>	
</configuration>	
</load-configuration>	
</rpc>	
	<rpc-reply></rpc-reply>
<rpc>	
<commit-configuration/>	
</rpc>	
	<rpc-reply></rpc-reply>
<rpc>	
<unlock-configuration/>	
</rpc>	
	<rpc-reply></rpc-reply>

T1004

The client application closes the JUNOScript session:

Client Application	JUNOScript Server
<rpc>	
<request-end-session/>	
</rpc>	
	<rpc-reply>
	<end-session/>
	</rpc-reply>
</junoscript>	
	</junoscript>

T1005

Chapter 3
Operational Requests

This chapter explains how to use the JUNOScript application programming interface (API) to request information about router status. The JUNOScript request tags described here correspond to JUNOS command-line interface (CLI) operational commands described in the *JUNOS Internet Software Operational Mode Command Reference*. The JUNOScript API defines a specific request tag for most commands in the CLI show family of commands. For example, the <get-interface-information> tag corresponds to the show interfaces command, and the <get-chassis-inventory> tag requests the same information as the show chassis command.

This chapter discusses the following topics:

For information about using the JUNOScript API to request router configuration information, see "Router Configuration" on page 39.

Request Operational Information

To request operational information from the JUNOScript server, first establish a connection to it and open a JUNOScript session as described in "Connect to the JUNOScript Server" on page 18 and "Start the JUNOScript Session" on page 21.

Then emit operational request tags, each enclosed in an opening <rpc> and closing </rpc> tag. Client applications can emit an unlimited number of operational request tags during a JUNOScript session (one at a time). It can freely intermingle them with configuration requests, which are described in "Router Configuration" on page 39.

Each operational request tag corresponds to a JUNOS CLI command that has a distinct function or returns a different kind of output. For a list of mappings between CLI commands and operational request tags, see the *JUNOScript API Reference*.

The JUNOScript API defines separate Extensible Markup Language (XML) document type definition (DTD) files for different JUNOS components. For instance, the DTD for interface information is called junos-interface.dtd and the DTD for chassis information is called junos-chassis.dtd. Each DTD constitutes a separate XML namespace, which means that multiple DTDs can define a tag with the same name but a distinct function. The *JUNOScript API Reference* includes the text of the JUNOScript DTDs.

Client applications can emit all tags that constitute a request by invoking one instance of a routine such as write(), or can invoke a separate instance of the routine for each tag or group of tags. The JUNOScript server ignores any newline characters, spaces, or other white space characters in the tag stream.

After the client application finishes making requests, it can close the JUNOScript session and terminate the connection as described in "End the Session and Close the Connection" on page 29.

JUNOS CLI commands take two main kinds of options, and there are matching child tags in the corresponding operational request tag. For a discussion and example of each, see the following sections:

- Map Child Tags to Options with Variable Values on page 36
- Map Child Tags to Fixed-Form Options on page 37

Map Child Tags to Options with Variable Values

Many JUNOS CLI commands have options that specify the name of the object that the command affects or reports about. In some cases, the CLI does not precede the option name with a fixed-form keyword, but XML convention requires that the JUNOScript API define a tag for each option. The tag is most often called <name>, but other names are sometimes used. To learn the names for an operational request tag's child tags, consult the tag's entry in the appropriate DTD or in the *JUNOScript API Reference*.

The following illustrates the mapping between CLI command and JUNOScript tags for two CLI operational commands with variable-form options. In the show interfaces command, t3-5/1/0:0 is the name of the interface. In the show bgp neighbor command, 10.168.1.222 is the IP address for the BGP peer of interest.

CLI Command	JUNOScript Tags
show interfaces t3-5/1/0:0	<rpc> <get-interface-information> <interface-name>t3-5:1/0:0</interface-name> </get-interface-information> </rpc>
show bgp neighbor 10.168.1.122	<rpc> <get-bgp-neighbor-information> <neighbor-address>10.168.1.122</neighbor-address> </get-bgp-neighbor-information> </rpc>

T1006

Map Child Tags to Fixed-Form Options

Some JUNOS CLI commands include options that have a fixed form, such as the brief and detail strings, which specify the amount of detail to include in the output. The JUNOScript API usually maps such an option to an empty tag whose name matches the option name.

The following illustrates the mapping between the CLI command and JUNOScript tags for the show isis adjacency command, which has a fixed-form option called detail:

CLI Command	JUNOScript Tags
show isis adjacency detail	<rpc> <get-isis-adjacency-information> <detail/> </get-isis-adjacency-information> </rpc>

T1007

Parse an Operational Response

The JUNOScript server encloses its response to each operational request in separate <rpc-reply> tags. For information about the conventions that client applications must follow when interpreting a JUNOScript server response, see “General JUNOScript Conventions” on page 9. For information about parsing a response tag, see “Parse the JUNOScript Server Response” on page 27.

Client applications must include code for accepting the tag stream and extracting the content from tags. They can, for instance, feed them to a parser that implements a standard API such as the Document Object Model (DOM) or Simple API for XML (SAX). For more information, see “Use a Standard API to Parse Response Tags” on page 29.

Requests and Responses without Defined JUNOScript Tags

The JUNOScript API does not define a tag for every JUNOS CLI command. Client applications can still invoke the functionality of such commands by enclosing the actual CLI command string in a <command> tag. Like all request tags, the <command> tag must occur within an <rpc> tag.

The following illustrates the use of the <command> tag to issue the request message command, for which the JUNOScript API does not define a tag. The line breaks in both the CLI command and within the <command> tags are for legibility only. The client application should not include newlines or other extraneous white space characters in the string within the <command> tags. Also, unlike most examples in this manual, the following does not preserve a line-to-line correspondence between the CLI command and the JUNOScript tags:

CLI Command	JUNOScript Tags
request message all message "Statistics gathering in progress"	<rpc> <command> request message all message "Statistics gathering in progress" </command> </rpc>

T1008

If the JUNOScript API does not define a response tag for the type of output requested by a client application, the JUNOScript server encloses its response in an <output> tag. The tag's contents are usually one or more lines of formatted ASCII output like that displayed by the JUNOS CLI on the computer screen.

The JUNOScript server might not support use of all CLI commands in the <command> tag. For details, see the JUNOS Internet software release notes.

The content and formatting of data within an <output> tag are subject to change, so client applications must not depend on them. Future versions of the JUNOScript API will define specific response tags (rather than <output> tags) for more commands. Client applications that rely on the content of <output> tags will not be able to interpret the output from future versions of the JUNOScript server.

Chapter 4
Router Configuration

This chapter explains how to use the JUNOScript application programming interface (API) to change router configuration or to request information about the current configuration. The JUNOScript tags described here correspond to JUNOS command-line interface (CLI) configuration statements described in the JUNOS Internet software configuration guides.

This chapter discusses the following topics:

Mapping between CLI Configuration Statements and JUNOScript Tags

The JUNOScript API defines a tag for every container and leaf statement in the JUNOS configuration hierarchy. At the top levels of the configuration hierarchy, there is almost always a one-to-one mapping between tags and statements, and most tag names match the configuration statement name. At deeper levels of the configuration hierarchy, the mapping is sometimes less direct, because some CLI notational conventions do not map directly to Extensible Markup Language (XML)-compliant tagging syntax. The following sections describe the mapping between configuration statements and JUNOScript tags:

For some configuration statements, the notation used when typing the statement at the CLI configuration-mode prompt differs from the notation used in a configuration file. The same JUNOScript tag maps to both notational styles.

Tag Mappings for Top-Level (Container) Statements

The <configuration> tag is the top-level JUNOScript container tag for configuration statements, and corresponds to the CLI [edit] level. Most statements at the next few levels of the configuration hierarchy are container statements, and the name of each corresponding JUNOScript container tag almost always matches the container statement name.

(The top-level <configuration-text> tag also corresponds to the CLI configuration mode's [edit] level. It encloses formatted ASCII configuration statements instead of JUNOScript tags, and is not relevant to the following discussion. For more information, see "Specify Formatted ASCII or JUNOScript-Tagged Output" on page 49.)

The following depicts the mappings for two sample statements that begin at the top level of the configuration hierarchy. Note how a closing brace in a CLI configuration statement corresponds to a closing JUNOScript tag:

CLI Configuration Statements

```
system {
    login {
        …child statements…
    }
}
protocols {
    ospf {
        …child statements…
    }
}
```

JUNOScript Tags

```
<configuration>
    <system>
        <login>
            <!- - child statements - ->
        </login>
    </system>
    <protocols>
        <ospf>
            <!- - child statements - ->
        </ospf>
    </protocols>
</configuration>
```

T1009

Tag Mappings for Leaf Statements

A *leaf statement* is a CLI configuration statement that does not contain any other statements. Most leaf statements define a value for one characteristic of a configuration object and have the following form:

keyword *value*;

In general, the name of the JUNOScript tag corresponding to a leaf statement is the same as the keyword string. The string between the opening and closing JUNOScript tags is the *value*.

The following depicts the mappings for two sample leaf statements that have a keyword and a value: the message statement at the [edit system login] level and the preference statement at the [edit protocols ospf] level:

CLI Configuration Statements

```
system {
    login {
        message "Authorized users only";
        …other statements under login …
    }
}
protocols {
    ospf {
        preference 15;
        …other statements under ospf …
    }
}
```

JUNOScript Tags

```
<configuration>
    <system>
        <login>
            <message>Authorized users only</message>
            <!- - other children of the <login> tag - ->
        </login>
    </system>
    <protocols>
        <ospf>
            <preference>15</preference>
            <!- - other children of the <ospf> tag - ->
        </ospf>
    </protocols>
</configuration>
```

T1010

Some leaf statements consist of a fixed-form keyword only, without an associated variable-form value. The JUNOScript API represents such statements with an empty tag. The following example shows the mapping for the disable statement at the [edit forwarding-options sampling] hierarchy level:

CLI Configuration Statement

```
forwarding-options {
    sampling {
        disable;
        …other statements under sampling …
    }
}
```

JUNOScript Tags

```
<configuration>
    <forwarding-options>
        <sampling>
            <disable/>
            <!- - other children of the <sampling> tag - ->
        </sampling>
    </forwarding-options>
</configuration>
```

T1011

Tag Mappings for Identifiers

At some hierarchy levels, the same kind of container object can appear multiple times. Each instance of the object has a unique identifier to distinguish it from the other instances. In the JUNOS CLI notation, the first statement in the set of statements for such an object consists of a keyword and identifier of the following form:

```
keyword identifier {
    … configuration statements for individual characteristics …
}
```

The keyword is a fixed string that indicates the type of object being defined, and the *identifier* is the unique name for this instance of the type. In the JUNOScript API, the tag corresponding to the keyword is a container tag for child tags that represent the object's characteristics. The container tag's name generally matches the keyword string.

The JUNOScript API differs from the CLI in its treatment of the identifier. Because the JUNOScript API does not allow container tags to contain both other tags and untagged character data such as an identifier name, the identifier must be enclosed in a tag of its own. Most frequently, identifier tags are called <name>. To verify the identifier tag name for an object, consult the object's entry in the appropriate document type definition (DTD) or in the *JUNOScript API Reference*.

Identifier tags also constitute an exception to the general XML convention that tags at the same level of hierarchy can appear in any order; the identifier tag always occurs first within the container tag.

The [edit protocols bgp group] statement is an example of a configuration statement with an identifier. Each Border Gateway Protocol (BGP) group has an associated name (the identifier) and can have other characteristics such as a type, peer autonomous system (AS) number, and neighbor address.

The following example illustrates the mapping between the CLI statements and JUNOScript tags that create two BGP groups called G1 and G2. Notice how the JUNOScript <name> tag that encloses the identifier for each group (and for the identifier of the neighbor within a group) does not have a counterpart in the CLI statements. For complete information about changing router configuration, see "Change the Candidate Configuration" on page 55.

CLI Configuration Statements

```
protocols {
   bgp {
      group G1 {

         type external;
         peer-as 56;
         neighbor 10.0.0.1;


      }
      group G2 {

         type external;
         peer-as 57;
         neighbor 10.0.10.1;


      }
   }
}
```

JUNOScript Tags

```
<configuration>
   <protocols>
      <bgp>
         <group>
            <name>G1</name>
            <type>external</type>
            <peer-as>56</peer-as>
            <neighbor>
               <name>10.0.0.1</name>
            </neighbor>
         </group>
         <group>
            <name>G2</name>
            <type>external</type>
            <peer-as>57</peer-as>
            <neighbor>
               <name>10.0.10.1</name>
            </neighbor>
         </group>
      </bgp>
   </protocols>
</configuration>
```

T1012

Tag Mappings for Leaf Statements with Multiple Values

Some JUNOS CLI leaf statements accept multiple values, which can either be user-defined or drawn from a set of predefined values. CLI notation uses square brackets to enclose all values in a single statement, as in the following:

statement [*value1 value2 value3*];

The JUNOScript API instead encloses each value in its own tag. The following example illustrates the mapping between a CLI statement with multiple user-defined values and the corresponding JUNOScript tags. The import statement imports two routing policies defined elsewhere in the configuration. For complete information about changing router configuration, see "Change the Candidate Configuration" on page 55.

CLI Configuration Statements

```
protocols {
   bgp {
      group 23 {

         import [ policy1 policy2 ];

      }
   }
}
```

JUNOScript Tags

```
<configuration>
   <protocols>
      <bgp>
         <group>
            <name>23</name>
            <import>policy1</import>
            <import>policy2</import>
         </group>
      </bgp>
   </protocols>
</configuration>
```

T1013

The following example illustrates the same mapping for a CLI statement with multiple predefined values. The permissions statement grants three predefined JUNOS permissions to members of the user-accounts login class.

CLI Configuration Statements

```
system {
    login {
        class user-accounts {

            permissions [ configure admin control ];



        }
    }
}
```

JUNOScript Tags

```
<configuration>
    <system>
        <login>
            <class>
                <name>user-accounts</name>
                <permissions>configure</permissions>
                <permissions>admin</permissions>
                <permissions>control</permissions>
            </class>
        </login>
    </system>
</configuration>
```

T1014

Tag Mappings for Multiple Options on One or More Lines

For some configuration objects, the configuration file specifies the value for more than one option on a single line, usually for greater legibility and conciseness. In most such cases, the first option identifies the object and does not have a keyword, but later options are paired keywords and values. The JUNOScript API encloses each option in its own tag. Because the first option has no keyword in the CLI statement, the JUNOScript API assigns a tag name to it.

The following example illustrates the mapping of a CLI configuration statement that places multiple options on a single line to the corresponding JUNOScript tags. Notice that the JUNOScript API defines a tag for both options and assigns a tag name to the first option (10.0.0.1), which has no CLI keyword.

CLI Configuration Statements

```
system {
    backup-router 10.0.01 destination 10.0.0.2;



}
```

JUNOScript Tags

```
<configuration>
    <system>
        <backup-router>
            <address>10.0.0.1</address>
            <destination>10.0.0.2</destination>
        </backup-router>
    </system>
</configuration>
```

T1015

The configuration file entries for some configuration objects have multiple lines with more than one option, and again the JUNOScript API defines a separate tag for each option. The following example illustrates the mappings for a sample [edit protocols isis traceoptions] statement, which contains three statements, each with multiple options:

CLI Configuration Statements

```
protocols {
   isis {
      traceoptions {
         file trace-file size 3m files 10 world-readable;

         flag route detail;

         flag state receive;

      }
   }
}
```

JUNOScript Tags

```
<configuration>
   <protocols>
      <isis>
         <traceoptions>
            <file>
               <filename>trace-file</filename>
               <size>3m</size>
               <files>10</files>
               <world-readable/>
            </file>
            <flag>
               <name>route</name>
               <detail/>
            </flag>
            <flag>
               <name>state</name>
               <receive/>
            </flag>
         </traceoptions>
      </isis>
   </protocols>
</configuration>
```

T1016

Same Tags Used for Requests and Responses

The JUNOScript server encloses its response to a configuration request in <rpc-reply> tags, just as for an operational request. Within the <rpc-reply> tags, it by default returns a JUNOScript-tagged response enclosed in <configuration> tags. (If the client application requests a formatted ASCII response, the server instead encloses the response in <configuration-text> tags. For more information, see “Specify Formatted ASCII or JUNOScript-Tagged Output” on page 49.)

Enclosing every JUNOScript-tagged configuration response within <configuration> tags contrasts with how the server encloses each different kind of operational response in a tag named for that type of response—for example, the <chassis-inventory> tag for chassis information or the <interface-information> tag for interface information.

The JUNOScript tags within the <configuration> tags represent configuration hierarchy levels, configuration objects, and object characteristics, always ordered from higher to deeper levels of the hierarchy. When a client application loads a configuration, it emits the same tags in the same order as the JUNOScript server uses when returning configuration information. This consistent representation helps make handling configuration information more straightforward. For instance, the client application can change router configuration by requesting the current configuration, storing the JUNOScript server’s response to a local memory buffer, making changes or applying transformations to the buffered data, and reloading the altered configuration. Because the altered configuration is based on the JUNOScript server’s response, it is certain to be syntactically correct. For more information about changing router configuration, see “Change the Candidate Configuration” on page 55.

Similarly, when a client application requests information about a configuration hierarchy level or object, it uses the same tags that the JUNOScript server will return in response. To represent the hierarchy level or object about which it is requesting information, the client application sends a complete stream of tags from the top of the configuration hierarchy (represented by the <configuration> tag) down to the requested level or object. The innermost tag, which represents the level or object, is either empty or includes the identifier tag only. The JUNOScript server's response includes the same stream of tags, but the tag for the requested level or object contains all the tags that represent the level's child tags or object's characteristics. For more information about requesting configuration information, see "Request Configuration Information" on page 48.

One potential difference between the tag stream in the JUNOScript server's response and one emitted by a client application concerns the use of white space. By XML convention, the JUNOScript server ignores white space in the tag stream it receives. In the stream that it emits, however, the JUNOScript server includes newline characters and extra spaces between tags. If a client application writes the response to a file, each tag appears on its own line, and child tags are indented from their parents, both of which enhance readability for users. Client applications can ignore or discard the white space, particularly if they do not write the tags to a file for later review. Client applications do not need to include the white space in requests to the JUNOScript server.

Overview of Router Configuration Procedures

To read or change router configuration information, perform the following procedures, which are described in the indicated sections:

1. Establish a connection to the JUNOScript server on the router, as described in "Prerequisites for telnet Connections" on page 17.

2. Open a JUNOScript session, as described in "Start the JUNOScript Session" on page 21.

3. (Optional) Lock the current candidate configuration, as described in "Lock the Candidate Configuration" on page 47. Locking the configuration prevents other users or applications from changing it at the same time.

4. Request or change configuration information, as described in "Request Configuration Information" on page 48 and "Change the Candidate Configuration" on page 55.

5. (Optional) Verify the syntactic correctness of the candidate configuration before attempting to commit it, as described in "Verify the Syntactic Correctness of the Candidate Configuration" on page 65.

6. Commit changes made to the candidate configuration (if appropriate), as described in "Commit the Candidate Configuration" on page 66.

7. Unlock the current configuration if it is locked, as described in "Unlock the Candidate Configuration" on page 68.

8. End the JUNOScript session and close the connection to the router, as described in "End the Session and Close the Connection" on page 29.

Lock the Candidate Configuration

Before reading or changing router configuration, a client application must establish a connection to the JUNOScript server, as described in “Prerequisites for telnet Connections” on page 17, and open a JUNOScript session, as described in “Start the JUNOScript Session” on page 21.

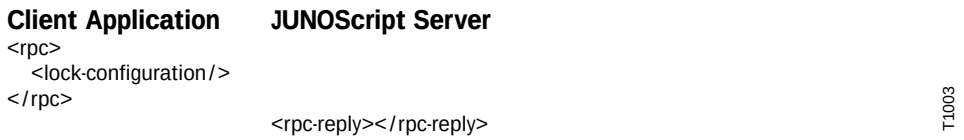
The application can then emit the <lock-configuration/> tag enclosed in <rpc> tags if it needs to prevent other users or applications from changing the candidate configuration. If it is not important to block changes from other applications, the application can begin requesting or changing configuration information immediately, as described in “Request Configuration Information” on page 48 and “Change the Candidate Configuration” on page 55.

Only one application can hold the lock on the candidate configuration at a time. Other users and applications can still read the configuration while it is locked. The lock persists until either the JUNOScript session ends or the client application emits the <unlock-configuration/> tag, as described in “Unlock the Candidate Configuration” on page 68.

If the JUNOScript server successfully locks the configuration, it returns an opening <rpc-reply> and closing </rpc-reply> tag with nothing between them. If it cannot lock the configuration, it returns an <xnm:error> tag instead. Reasons for the failure include the following:

- Another user or application has already locked the candidate configuration. The error message reports the login identity of the user or application.
- The candidate configuration includes changes that have not yet been committed. To commit the changes, see “Commit the Candidate Configuration” on page 66. To roll back to a previous version of the configuration (and lose the uncommitted changes), see “Roll Back to a Previous Configuration” on page 65.

In the following example, the client application emits the <lock-configuration/> tag after opening the JUNOScript session and exchanging initialization information with the JUNOScript server:



Automatically Discard Uncommitted Changes

By default, if a client application locks the configuration and does not commit it before the JUNOScript session ends, any uncommitted changes that are made during the session are retained in the candidate configuration. When the candidate configuration is subsequently committed, the leftover changes become part of the committed configuration. This can lead to unexpected results if the user or application performing the subsequent commit is unaware of the leftover changes.

To discard uncommitted changes from the candidate configuration when the JUNOScript session ends before the client application commits the configuration, enclose the <rollback> tag within the <lock-configuration> tag and set the <rollback> tag's value to automatic.

The following example illustrates the sequence of tags that causes an automatic rollback:

Client Application	JUNOScript Server
<rpc> <lock-configuration> <rollback>automatic</rollback> </lock-configuration> </rpc>	
	<rpc-reply></rpc-reply>

T1017

Request Configuration Information

To request and parse configuration information, a client application emits the <get-configuration> tag. By setting optional attributes on the <get-configuration> tag and enclosing the appropriate child tags within it, the client application can request either the candidate or the committed configuration, specify either JUNOScript-tagged or formatted ASCII output, and request the entire configuration or just a section of it. The following sections describe the procedures for requesting configuration information:

- Specify the Committed or Candidate Configuration on page 49
- Specify Formatted ASCII or JUNOScript-Tagged Output on page 49
- Request the Complete Configuration on page 51
- Request One Hierarchy Level on page 51
- Request a Single Configuration Object on page 52

If the client application has locked the candidate configuration as described in "Lock the Candidate Configuration" on page 47, it should unlock it after making its read requests. Other users and applications cannot change the configuration while it remains locked. For more information, see "Unlock the Candidate Configuration" on page 68.

Client applications can also request an XML schema representation of the complete hierarchy of JUNOS configuration statements, as described in the following section:

- Request an XML Schema for the Configuration Hierarchy on page 53

Specify the Committed or Candidate Configuration

To request information from the current committed configuration—the one active on the router—set the database attribute on the <get-configuration> tag to the value committed. To request information from the current candidate configuration, either set the database attribute to the value candidate or omit the database attribute completely (the JUNOScript server returns information from the candidate configuration by default). In either case, enclose the <get-configuration> tag in <rpc> tags.

The database attribute can be combined with the format attribute (described in "Specify Formatted ASCII or JUNOScript-Tagged Output" on page 49), and included when requesting either the entire configuration or sections of it (as described in "Request the Complete Configuration" on page 51, "Request One Hierarchy Level" on page 51, and "Request a Single Configuration Object" on page 52).

The following example uses the database attribute to request the entire committed configuration:

Client Application

```
<rpc>
   <get-configuration database="committed"/>
</rpc>
```

JUNOScript Server

```
<rpc-reply>
   <configuration>
      <version>5.3R1</version>
         <system>
            <host-name>big-router</host-name>
            <!- - other children of the <system> tag - ->
         </system>
      <!- - other children of the <configuration> tag - ->
   </configuration>
</rpc-reply>
```

T1018

Specify Formatted ASCII or JUNOScript-Tagged Output

To request that the JUNOScript server return configuration data as formatted ASCII text rather than tagged with JUNOScript tags, set the format attribute on the <get-configuration> tag to the value text. The server formats its response in the same way as the JUNOS CLI show configuration command displays configuration data—it uses the newline character, tabs, braces, and square brackets to indicate the hierarchical relationships between configuration statements. The server encloses formatted ASCII configuration information in <configuration-text> tags.

To request JUNOScript-tagged output, either set the format attribute to the value xml or omit the format attribute completely (the JUNOScript server returns JUNOScript-tagged output by default). The JUNOScript server encloses its output in <configuration> tags.

When the JUNOScript server encloses a JUNOScript tag in <undocumented> tags, the corresponding configuration element (hierarchy level or object) is not documented in the JUNOS software configuration guides or officially supported by Juniper Networks. Most often, the undocumented element is used for debugging purposes only by Juniper Networks personnel. In rarer cases, the element is no longer supported or has been moved to another area of the configuration hierarchy, but appears in the current location for backward compatibility.

Regardless of whether they are requesting JUNOScript tags for formatted ASCII, client applications must use JUNOScript tags to represent the configuration element to display. The format attribute controls the format of only the JUNOScript server's output. Enclose the tags that represent the element to display in <get-configuration> tags within <rpc> tags.

The format attribute can be combined with the database attribute (described in "Specify the Committed or Candidate Configuration" on page 49), and included when requesting either the entire configuration or sections of it (as described in "Request the Complete Configuration" on page 51, "Request One Hierarchy Level" on page 51, and "Request a Single Configuration Object" on page 52).

The following example uses the format attribute to request ASCII-formatted output at the [edit policy-options] level of the current candidate configuration:

Client Application

```
<rpc>
   <get-configuration format="text">
      <configuration>
         <policy-options/>
      </configuration>
   </get-configuration>
</rpc>
```

JUNOScript Server

```
<rpc-reply>
   <configuration-text>
      policy-options {
         policy-statement load-balancing-policy {
            from {
               route-filter 192.168.10/24 orlonger;
               route-filter 9.114/16 orlonger;
            }
            then {
               load-balance per-packet;
            }
         }
      }
   </configuration-text>
</rpc-reply>
```

T1019

Request the Complete Configuration

To request the entire current committed or candidate configuration, emit the empty <get-configuration/> tag enclosed in <rpc> tags. If desired, include the database or format attribute (or both), as described in "Specify the Committed or Candidate Configuration" on page 49 and "Specify Formatted ASCII or JUNOScript-Tagged Output" on page 49, respectively.

The following example illustrates the tag sequence for requesting the current candidate configuration tagged with JUNOScript tags (the default):

Client Application

```
<rpc>
   <get-configuration/>
</rpc>
```

JUNOScript Server

```
<rpc-reply>
   <configuration>
      <version>5.3R1</version>
      <system>
         <host-name>big-router</host-name>
         <!- - other children of the <system> tag - ->
      </system>
      <!- - other children of the <configuration> tag - ->
   </configuration>
</rpc-reply>
```

T1020

Request One Hierarchy Level

To request a single hierarchy level from the current committed or candidate configuration, emit <get-configuration> tags that enclose the tags representing all levels of the configuration hierarchy from the root (represented by the <configuration> tag) down to the level to display. Use an empty tag to represent the requested level. Enclose the entire request in <rpc> tags.

If desired, include the database or format attribute (or both), as described in "Specify the Committed or Candidate Configuration" on page 49 and "Specify Formatted ASCII or JUNOScript-Tagged Output" on page 49, respectively. Note, however, that the format attribute controls the format of only the JUNOScript server's output. Client applications must emit JUNOScript tags rather than formatted ASCII to represent which configuration level to display.

The following example illustrates the tag sequence when a client application requests the contents of the [edit system login] level of the current candidate configuration. The output is tagged with JUNOScript tags (the default):

Client Application

```
<rpc>
   <get-configuration>
      <configuration>
         <system>
            <login/>
         </system>
      </configuration>
   </get-configuration>
</rpc>
```

JUNOScript Server

```
<rpc-reply>
   <configuration>
      <system>
         <login>
            <user>
               <name>barbara</name>
               <full-name>Barbara Anderson</full-name>
               <!- - other child tags for this user - ->
            </user>
            <!- - other children of the <login> tag - ->
         </login>
      </system>
   </configuration>
</rpc-reply>
```

T1021

Request a Single Configuration Object

To request information about a single object at a specific level of the configuration hierarchy, emit <get-configuration> tags that enclose the tags representing the entire configuration hierarchy down to the object to display. To represent the requested object, emit its container tag and identifier tag only, not any tags that represent other characteristics.

If desired, include the database or format attribute (or both), as described in “Specify the Committed or Candidate Configuration” on page 49 and “Specify Formatted ASCII or JUNOScript-Tagged Output” on page 49, respectively. Note, however, that the format attribute controls the format of only the JUNOScript server’s output. Client applications must emit JUNOScript tags rather than formatted ASCII to represent which configuration object to display.

The following example illustrates the tag sequence when a client application requests the contents of one multicasting scope called local, which is at the [edit routing-options multicast] hierarchy level. To specify the configuration object about which to supply information, the client application emits the <name>local</name> identifier tag as the innermost tag. The output is from the current candidate configuration and is tagged with JUNOScript tags (the default).

Client Application

```
<rpc>
   <get-configuration>
      <configuration>
         <routing-options>
            <multicast>
               <scope>
                  <name>local</name>
               </scope>
            </multicast>
         </routing-options>
      </configuration>
   </get-configuration>
</rpc>
```

JUNOScript Server

```
<rpc-reply>
   <configuration>
      <routing-options>
         <multicast>
            <scope>
               <name>local</name>
               <prefix>239.255.0.0/16</prefix>
               <interface>ip-f/p/0</interface>
            </scope>
         </multicast>
      </routing-options>
   </configuration>
</rpc-reply>
```

T1022

Request an XML Schema for the Configuration Hierarchy

To request an XML Schema-language representation of the entire JUNOS configuration hierarchy, emit <get-xnm-information> tags that enclose the following two child tags with the indicated values:

- <type>xml-schema</type>, which specifies that the JUNOScript server return the configuration as an XML schema
- <namespace>junos-configuration</namespace>, which specifies that the JUNOScript server return information about the JUNOS configuration

The JUNOScript server returns the XML schema enclosed in <rpc-reply> and <xsd:schema> tags. The JUNOScript configuration schema represents the entire set of elements that can be configured on a Juniper Networks router that is running the version of the JUNOS software specified by the xmlns:junos attribute in the opening <rpc-reply> tag. Client applications can use the schema to validate the candidate or committed schema on a router, or to discover which configuration statements are available in that version of the JUNOS software.

The JUNOScript configuration schema does not indicate which elements are actually configured on the router where the JUNOScript server is running, or even that an element can be configured on that type of router (some configuration statements are available only on certain router types). To request the set of currently configured elements and their settings, emit the <get-configuration> tag instead, as described in other sections in this chapter.

Explaining the structure and notational conventions of the XML Schema language is beyond the scope of this document. For a basic introduction to the XML Schema language, see the primer available at http://www.w3.org/TR/xmlschema-0. The primer references the more formal specifications for the XML Schema language if you need additional details.

The following examples illustrate the tag sequence with which a client application requests the JUNOScript configuration schema. In the JUNOScript server's reply, the first two <xsd:element> statements define the schema for the <undocumented> and <comment> JUNOScript tags, which can be enclosed in most other container tags defined in the schema (container tags are defined as <xsd:complexType>).

Client Application

```
<rpc>
   <get-xnm-information>
      <type>
         xml-schema
      </type>
      <namespace>
         junos-configuration
      </namespace>
</rpc>
```

JUNOScript Server

```
<rpc-reply>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" \
                                        elementFormDefault="qualified">
   <xsd:element name="undocumented">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:any namespace="##any" processContents="skip"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="comment">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:any namespace="##any" processContents="skip"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
         .
         .
         .
```

T1023

The third <xsd:element> statement in the schema defines the JUNOScript <configuration> tag. The fourth <xsd:element> statement begins the definition of the <system> tag, which corresponds to the [edit system] level of the JUNOScript configuration hierarchy. The statements corresponding to other hierarchy levels are omitted for brevity.

Client Application JUNOScript Server

```
                .
                .
                .
        </xsd:element>
          <xsd:element name="configuration">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:choice minOccurs="0" maxOccurs="unbounded">
                  <xsd:element ref="undocumented"/>
                  <xsd:element ref="comment"/>
                  <xsd:element name="system" minOccurs="0">
                    <xsd:complexType>
                      <xsd:sequence>
                        <xsd:choice minOccurs="0" maxOccurs="unbounded">
                          <xsd:element ref="undocumented"/>
                          <xsd:element ref="comment"/>
                          <!- - child elements of <system> appear here - ->
                        </xsd:choice >
                      </xsd:sequence>
                    </xsd:complexType>
                  </xsd:element>
                  <!- - statements for other hierarchy levels appear here - ->
                </xsd:choice >
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
        </xsd:schema>
      </rpc-reply>
```

T1024

Change the Candidate Configuration

To change the current candidate configuration, emit the <load-configuration> tag. Specify which configuration element (hierarchy level or configuration object) to change in one of two ways:

- By setting the empty <load-configuration> tag's url attribute to the pathname of a file that resides on the router and contains the set of configuration statements to load. For example, the following tag identifies the file /tmp/new.conf as the file to load:

  ```
  <load-configuration url="/tmp/new.conf"/>
  ```

- By enclosing the appropriate configuration statements within opening and closing <load-configuration> tags. Include the complete statement path down to the element to change.

To confirm that it successfully loaded the configuration as the new candidate configuration, the JUNOScript server returns <rpc-reply></rpc-reply> tags with nothing between them. If it cannot load the configuration, it emits an <xnm:error> tag instead.

Client applications can provide the configuration information to load either as formatted ASCII or tagged with JUNOScript tags. They can also specify the manner in which the JUNOScript server loads the configuration. For instructions, see the following sections:

Provide Configuration Data as Formatted ASCII or JUNOScript Tags

Client applications can use one of two formats when providing configuration data to be loaded into the candidate configuration:

- If providing formatted ASCII (the standard format used by the JUNOS CLI show configuration command to display configuration data), set the format attribute on the <load-configuration> tag to the value text. Enclose the configuration data in <configuration-text> rather than <configuration> tags. Format the configuration information to be loaded by using the newline character, tabs and other white space, braces, and square brackets to indicate the hierarchical relationships between the configuration statements.

- If providing JUNOScript-tagged information, either set the format attribute on the <load-configuration> tag to the value xml or omit the format attribute completely (the JUNOScript server expects JUNOScript-tagged output by default). Enclose the configuration information in <configuration> tags.

Whichever form of configuration information is provided, enclose the <load-configuration> tag in <rpc> tags. The format attribute can be combined with the action attribute described in "Merge Statements into the Current Configuration" on page 56, "Replace (Override) the Entire Current Configuration" on page 58, and "Replace a Configuration Element" on page 58. For examples of the possible combinations, see those sections.

Merge Statements into the Current Configuration

To combine the statements in the loaded configuration with the current candidate configuration, set the action attribute on the opening <load-configuration> tag to the value merge. This is also the default behavior if there is no action attribute.

```
<load-configuration action="merge">
```

Merging configuration statements is useful when adding a new configuration object or subhierarchy to the configuration. If statements in the loaded configuration conflict with statements in the current candidate configuration, the loaded statements replace the current ones.

As noted in "Provide Configuration Data as Formatted ASCII or JUNOScript Tags" on page 56, client applications can specify the configuration information to load either as formatted ASCII or tagged with JUNOScript tags. In the former case, set the format attribute on the <load-configuration> tag to text.

The following example merges information for a new interface called so-3/0/0 into the [edit interfaces] level of the current candidate configuration. The information is provided in JUNOScript-tagged format (the default).

Client Application **JUNOScript Server**

```
<rpc>
   <load-configuration action="merge">
      <configuration>
         <interfaces>
            <interface>
               <name>so-3/0/0</name>
               <unit>
                  <name>0</name>
                  <family>
                     <inet>
                        <address>
                           <name>10.0.0.1/8</name>
                        </address>
                     </inet>
                  </family>
               </unit>
            </interface>
         </interfaces>
      </configuration>
   </load-configuration>
</rpc>
```

```
<rpc-reply></rpc-reply>
```

T1025

The following example uses formatted ASCII to define the same new interface:

Client Application **JUNOScript Server**

```
<rpc>
   <load-configuration action="merge" format="text">
      <configuration-text>
         interfaces {
            so-3/0/0 {
               unit 0 {
                  family inet {
                     address 10.0.0.1/8;
                  }
               }
            }
         }
      </configuration-text>
   </load-configuration>
</rpc>
```

```
<rpc-reply></rpc-reply>
```

T1026

Replace (Override) the Entire Current Configuration

To discard the entire current candidate configuration and replace it with the loaded configuration, set the action attribute on the opening <load-configuration> tag to the value override:

```
<load-configuration action="override">
```

In the following example, the contents of the file /tmp/new.conf (which resides on the router) replaces the entire current candidate configuration. The information in the file is tagged with JUNOScript tags (the default), so the format attribute is not set.

Client Application

```
<rpc>
   <load-configuration action="override" url="/tmp/new.conf"/>
</rpc>
```

JUNOScript Server

```
<rpc-reply></rpc-reply>
```

T1027

Replace a Configuration Element

To replace a configuration element (hierarchy level or configuration object) in the current configuration, set the action attribute on the opening <load-configuration> tag to the value replace:

```
<load-configuration action="replace">
```

If using JUNOScript tags to define the loaded configuration, include the tags that represent the entire hierarchy down to the element you want to replace. In the opening container tag that represents the element, set the replace attribute to the value replace. Within the element's container tags, include all its child tags.

If using formatted ASCII to define the loaded configuration, include the statements that represent the entire hierarchy down to the element you want to replace. Place the replace: statement above the element's parent statement. Within the container tags for the element, include all relevant child statements.

The following example grants new permissions for the object named operator at the [edit system login class] hierarchy level. The information is provided in JUNOScript-tagged format (the default).

Client Application **JUNOScript Server**

```
<rpc>
   <load-configuration action="replace">
      <configuration>
         <system>
            <login>
               <class replace="replace">
                  <name>operator</name>
                  <permissions>configure</permissions>
                  <permissions>admin-control</permissions>
               </class>
            </login>
         </system>
      </configuration>
   </load-configuration>
</rpc>
```

```
<rpc-reply></rpc-reply>
```

T1028

The following example uses formatted ASCII to make the same change:

Client Application **JUNOScript Server**

```
<rpc>
   <load-configuration action="replace" format="text">
      <configuration-text>
         system {
            login {
            replace:
               class operator {
                  permissions [ configure admin-control ];
               }
            }
         }
      </configuration-text>
   </load-configuration>
</rpc>
```

```
<rpc-reply></rpc-reply>
```

T1029

Delete a Configuration Element

The client application can use the delete attribute to delete several kinds of configuration elements:

- Delete a Hierarchy Level on page 60
- Delete a Configuration Object on page 60
- Delete One or More Values from a Leaf Statement on page 61
- Delete a Fixed-Form Option on page 62

> **Note** The JUNOS CLI does not provide a delete: statement that marks formatted ASCII configuration data for deletion. Client applications must use JUNOScript tags to represent the data to delete.

Delete a Hierarchy Level

To remove a hierarchy level from the configuration hierarchy, set the delete attribute to the value delete on the empty tag that represents the level. Emit <load-configuration> tags that enclose the tags representing the entire statement path down to the level to remove.

The following example removes the [edit protocols ospf] level of the current candidate configuration:

Client Application

```
<rpc>
   <load-configuration>
      <configuration>
         <protocols>
            <ospf delete="delete"/>
         </protocols>
      </configuration>
   </load-configuration>
</rpc>
```

JUNOScript Server

```
<rpc-reply></rpc-reply>
```

T1030

Delete a Configuration Object

To delete a single configuration object, set the delete attribute to the value delete on the container tag for that object. Inside the container tag, include the identifier tag only, not any tags that represent other characteristics. Emit <load-configuration> tags that enclose the tags representing the entire statement path down to the object to remove.

> **Note** The delete attribute is placed on the containing tag rather than the identifier (in the following example, on the <user> tag rather than the <name> tag). The presence of the identifier tag results in the removal of the specified object rather than the removal of the entire hierarchy level represented by the containing tag (in the example, the user account barbara is removed rather than the entire [edit system login user] level).

The following example removes the account for user object barbara from the [edit system login user] level of the current candidate configuration:

Client Application

```
<rpc>
   <load-configuration>
      <configuration>
         <system>
            <login>
               <user delete="delete">
                  <name>barbara</name>
               </user>
            </login>
         </system>
      </configuration>
   </load-configuration>
</rpc>
```

JUNOScript Server

```
<rpc-reply></rpc-reply>
```

T1031

Delete One or More Values from a Leaf Statement

To delete one or more values from a leaf statement that accepts multiple values, set the delete attribute to the value delete on the tag for each value. Do not include the tags that represent values to be retained. Emit <load-configuration> tags that enclose the tags representing the entire statement path down to the leaf statement from which to remove values.

The following example removes two of the permissions granted to the user-accounts login class:

Client Application

```
<rpc>
   <load-configuration>
      <configuration>
         <system>
            <login>
               <class>
                  <name>user-accounts</name>
                  <permissions delete="delete">configure</permissions>
                  <permissions delete="delete">control</permissions>
               </class>
            </login>
         </system>
      </configuration>
   </load-configuration>
</rpc>
```

JUNOScript Server

```
<rpc-reply></rpc-reply>
```

T1032

Delete a Fixed-Form Option

To delete a fixed-form option, set the delete attribute to the value delete on the tag for the option. Emit <load-configuration> tags that enclose the tags representing the entire statement path down to the container tag for the option to remove.

The following example removes the fixed-form disable option at the [edit forwarding-options sampling] hierarchy level:

Client Application

```
<rpc>
   <load-configuration>
      <configuration>
         <forwarding-options>
            <sampling>
               <disable delete="delete"/>
            </sampling>
         </forwarding-options>
      </configuration>
   </load-configuration>
</rpc>
```

JUNOScript Server

```
<rpc-reply></rpc-reply>
```

T1033

Change a Configuration Element's Activation State

When a configuration element (hierarchy level or configuration object) is deactivated in the configuration hierarchy, it remains in the candidate configuration but is not activated in the actual configuration when the candidate is later committed.

To use JUNOScript tags to define which element to deactivate, either omit the format attribute from the opening <load-configuration> tag or set it to the value xml. Emit the <configuration> tag next, and then the tags representing the entire statement path down to the element to deactivate. On the tag that represents the element itself, set the inactive attribute to the value inactive. To represent a hierarchy level, emit an empty tag. To represent an object, emit the object's container tag. Inside the container tag enclose only the identifier tag for the object, not any tags that represent other object attributes.

To use formatted ASCII to define the element to deactivate, set the format attribute on the opening <load-configuration> tag to the value text. Emit the <configuration-text> tag next, and within it provide formatted ASCII for all statements in the path down to the element you want to deactivate. Place the inactive: statement immediately before the statement for the element.

To reactivate an element that was previously deactivated, use the preceding instructions and substitute the string active for inactive. Specifically, when loading JUNOScript tags, set the active attribute to the value active on the tag that represents the element to activate. When loading formatted ASCII statements, place the active: statement immediately before the statements to reactivate. With both kinds of notation, the reactivated element is activated in the committed configuration the next time the candidate configuration is committed.

The following example deactivates the [edit protocols isis] level of the current candidate configuration:

Client Application

```
<rpc>
   <load-configuration>
      <configuration>
         <protocols>
            <isis inactive="inactive">
         </protocols>
      </configuration>
   </load-configuration>
</rpc>
```

JUNOScript Server

```
<rpc-reply></rpc-reply>
```

T1034

Replace a Configuration Element and Change Its Activation State Simultaneously

To replace a configuration element (hierarchy level or configuration object) completely while simultaneously deactivating or reactivating it, set the action attribute on the enclosing <load-configuration> tag to the value replace. Within the container tags for the element you are replacing, include the tags or statements that represent all the element's characteristics, not just its identifier tag or statement. Finally, combine attributes or statements as follows:

- If using JUNOScript tags to replace and deactivate an element, set two attributes on its container tag: the replace attribute to the value replace and the inactive attribute to the value inactive. If using formatted ASCII, place the replace: statement above the statements to replace and the inactive: statement directly in front of the first statement.

- If using JUNOScript tags to replace and reactivate an element, set two attributes on its container tag: the replace attribute to the value replace and the active attribute to the value active. If using formatted ASCII, place the replace: statement above the statements to replace and the active: statement directly in front of the first statement.

For more information about completely reconfiguring an element, see "Replace a Configuration Element" on page 58.

The following example replaces the information in the [edit protocols bgp] level of the current candidate configuration for the group called G3, but also deactivates the group so that it is not activated in the actual configuration when the candidate is committed:

Client Application **JUNOScript Server**

```
<rpc>
   <load-configuration action="replace">
      <configuration>
         <protocols>
            <bgp>
               <group replace="replace" inactive="inactive">
                  <name>G3</name>
                  <type>external</type>
                  <peer-as>58</peer-as>
                  <neighbor>
                     <name>10.0.20.1</name>
                  </neighbor>
               </group>
            </bgp>
         </protocols>
      </configuration>
   </load-configuration>
</rpc>
```

```
<rpc-reply></rpc-reply>
```

T1035

The following example uses formatted ASCII to make the same change:

Client Application **JUNOScript Server**

```
<rpc>
   <load-configuration action="replace" format="text">
      <configuration-text>
         protocols {
            bgp {
            replace:
               inactive: group G3 {
                  type external;
                  peer-as 58;
                  neighbor 10.0.20.1;
               }
            }
         }
      </configuration-text>
   </load-configuration>
</rpc>
```

```
<rpc-reply></rpc-reply>
```

T1036

Roll Back to a Previous Configuration

The router stores a copy of the most recently committed configuration and up to nine previous configurations. To replace the current candidate configuration with a previous one, set the rollback attribute on the empty <load-configuration/> tag to the numerical index for the appropriate previous configuration. The index for the most recently committed configuration is 0 (zero), and the index for the oldest possible stored configuration is 9.

In the following example, the current candidate configuration is replaced by the most recently committed one:

Client Application

```
<rpc>
   <load-configuration rollback="0"/>
</rpc>
```

JUNOScript Server

```
<rpc-reply></rpc-reply>
```

T1037

Verify the Syntactic Correctness of the Candidate Configuration

Before committing the candidate configuration, you might want to confirm that it is syntactically correct. To verify the candidate configuration without actually committing it, enclose the empty <check/> tag in <commit-configuration> tags.

The JUNOScript server encloses its response in <rpc-reply> tags. If the configuration is incorrect, the enclosed <xnm:error> tag explains the nature of the error. The absence of error messages indicates that the operation succeeded.

The following example illustrates the tag sequence when the candidate configuration is syntactically correct:

Client Application

```
<rpc>
   <commit-configuration>
      <check/>
   </commit-configuration>
</rpc>
```

JUNOScript Server

```
<rpc-reply>
   <output>configuration check succeeds</output>
</rpc-reply>
```

T1038

Although the JUNOScript server currently indicates success by returning the <output> tag sequence shown in the example, future versions of the JUNOScript API might indicate success with a different string, a different tag, or no tag at all within the <rpc-reply> tag. Client applications must not rely on the presence or content of <output> tags, but instead should interpret the absence of error messages as an indicator of success.

Commit the Candidate Configuration

To commit the current candidate configuration so that it becomes the active configuration on the router, enclose the empty <commit-configuration/> tag in <rpc> tags. To avoid inadvertently committing changes made by other users or applications, lock the candidate configuration before changing it and emit the <commit-configuration/> tag while the configuration is still locked. (For instructions on locking and changing the candidate configuration, see "Lock the Candidate Configuration" on page 47 and "Change the Candidate Configuration" on page 55.) After committing the configuration, unlock it as described in "Unlock the Candidate Configuration" on page 68.

The JUNOScript server encloses its response in <rpc-reply> tags. If it cannot commit the configuration, the enclosed <xnm:error> tag explains the nature of the error. The most common causes of failure are semantic or syntactic errors in the candidate configuration. The absence of error messages within the <rpc-reply> tags indicates that the operation succeeded.

The following example illustrates the tag sequence when a client application commits the candidate configuration:

Client Application

```
<rpc>
   <commit-configuration/>
</rpc>
```

JUNOScript Server

```
<rpc-reply>
   <output>commit complete</output>
</rpc-reply>
```

T1039

Although the JUNOScript server currently indicates success by returning the <output> tag sequence shown in the example, future versions of the JUNOScript API might indicate success with a different string, a different tag, or no tag at all within the <rpc-reply> tag. Client applications must not rely on the presence or content of <output> tags, but instead should interpret the absence of error messages as an indicator of success.

Commit a Configuration but Require Confirmation

To commit the current candidate configuration but require an explicit confirmation for the commit to become permanent, emit the empty <confirmed/> tag enclosed in <commit-configuration> tags. If the commit is not confirmed within a certain amount of time (10 minutes by default), the JUNOScript server automatically rolls back to the previously committed configuration. To specify a different number of minutes for the rollback deadline, also emit the <confirm-timeout> tag and enclose an integer value.

The JUNOScript server encloses its response in <rpc-reply> tags. If it cannot commit the configuration, the enclosed <xnm:error> tag explains the nature of the error. The most common causes of commitment failure are semantic or syntactic errors in the candidate configuration. The absence of error messages within the <rpc-reply> tags indicates that the operation succeeded.

The <confirmed/> tag is useful for verifying that a configuration change works correctly and does not prevent management access to the router. If the change prevents access or causes other errors, the automatic rollback to the previous configuration restores access after the rollback deadline passes.

To delay the rollback to a time later than the current rollback deadline, emit the <confirmed/> tag again (enclosed in <commit-configuration> tags) before the deadline passes. Optionally, include the <confirm-timeout> tag to specify how long to delay the next rollback; omit that tag to delay the rollback by the default 10 minutes. The client application can delay the rollback indefinitely by emitting the <confirmed/> tag repeatedly in this way.

To cancel the rollback completely (and commit a configuration permanently), emit one of the following tag sequences before the rollback deadline passes:

- The empty <commit-configuration/> tag. The JUNOScript server commits the current candidate configuration immediately and cancels the rollback. If the candidate configuration is still the same as the temporarily committed configuration, this effectively recommits the temporarily committed configuration. The JUNOScript server confirms the commitment by returning an opening <rpc-reply> and closing </rpc-reply> tag with nothing between them.

- The empty <check/> tag enclosed in <commit-configuration> tags. The JUNOScript server confirms the syntactic correctness of the candidate configuration and cancels the rollback. The current committed configuration becomes permanently rather than temporarily committed. The JUNOScript server confirms that the candidate configuration is correct by returning an opening <rpc-reply> and closing </rpc-reply> tag with nothing between them.

The following example illustrates the tag sequence when a client application commits the candidate configuration with a rollback deadline of 20 minutes:

Client Application

```
<rpc>
   <commit-configuration>
      <confirmed/>
      <confirm-timeout>20</confirm-timeout>
   </commit-configuration>
</rpc>
```

JUNOScript Server

```
<rpc-reply>
   <output>commit complete</output>
</rpc-reply>
```

T1040

Although the JUNOScript server currently indicates success by returning the <output> tag sequence shown in the example, future versions of the JUNOScript API might indicate success with a different string, a different tag, or no tag at all within the <rpc-reply> tag. Client applications must not rely on the presence or content of <output> tags, but instead should interpret the absence of error messages as an indicator of success.

Unlock the Candidate Configuration

To unlock the candidate configuration after changing it, committing it, or both, emit the empty <unlock-configuration/> tag enclosed in <rpc> tags. Other applications and users cannot change the candidate configuration until the client application releases the lock. To confirm that it has successfully unlocked the configuration, the JUNOScript server returns an opening <rpc-reply> and closing </rpc-reply> tag with nothing between them. If it cannot unlock the configuration, it returns an <xnm:error> tag instead.

The following example illustrates the tag sequence with which the client application unlocks the configuration:

Client Application	JUNOScript Server
<rpc> <unlock-configuration/> </rpc>	
	<rpc-reply></rpc-reply>

T1041

Part 3

Write JUNOScript Client Applications

Chapter 5

Write a Perl Client Application

Juniper Networks provides a Perl module, called JUNOS::Device, to help you more quickly and easily develop custom Perl scripts for configuring and monitoring routers. The module implements an object that client applications can use to communicate with the JUNOScript server on a router. Accompanying the JUNOS::Device module are several sample Perl scripts, which illustrate how to use the module in scripts that perform various functions.

This chapter discusses the following topics:

Download the Module and Sample Scripts

To download, uncompress, and unpack the compressed tar-format file that contains the JUNOS::Device module and sample scripts, perform the following steps:

1. Access the Juniper Networks Customer Support Center Web page at http://www.juniper.net/support.
2. Click on the link labeled “JUNOScript API Software.”
3. For installation instructions, click on the link to the README.html file in the “JUNOScript Perl Module and Sample Scripts” section, and see the “Installation” section of the README.html file.

Module and Sample Scripts

The JUNOScript Perl distribution uses the directory structure for Perl modules that is used by the Comprehensive Perl Archive Network (http://www.cpan.org). There is a lib directory for the JUNOS::Device module and its supporting files, and an examples directory for the sample scripts.

The JUNOS::Device Perl module implements an object that client applications can use to communicate with a JUNOScript server; all the sample scripts use it.

The sample scripts illustrate how to perform the following functions:

- diagnose_bgp.pl—Illustrates how to write scripts to monitor router status and diagnose problems. The sample script extracts and displays information about a router's unestablished Border Gateway Protocol (BGP) peers from the full set of BGP configuration data.

- get_chassis_inventory.pl—Illustrates how to use one of the predefined Perl JUNOScript queries to request information from a router. The sample script invokes the get_chassis_inventory query, which requests the same information as the <get-chassis-inventory> JUNOScript tag and the JUNOS command-line interface (CLI) show chassis hardware command. For a list of all Perl queries available in this release of JUNOScript, see Table 6.

- load_configuration.pl—Illustrates how to change the router configuration by loading a file of configuration data that is formatted with JUNOScript tags. The distribution includes two sample files, set_login_user_foo.xml and set_login_class_bar.xml, but you can specify another JUNOScript configuration file on the command line.

The following sample scripts are used together to illustrate how to store and retrieve JUNOScript (or any Extensible Markup Language [XML]) data in a relational database. The scripts create and manipulate MySQL tables, but illustrate data manipulation techniques that apply to any relational database:

- get_config.pl—Illustrates how to retrieve router configuration information.

- make_tables.pl—Generates a set of Structure Query Language (SQL) statements for creating relational database tables and inserting data extracted from a specified XML file.

- pop_tables.pl—Populates existing relational database tables with data extracted from a specified XML file.

- unpop_tables.pl—Transforms data stored in a relational database table into XML and writes it to a file.

For instructions on running the scripts, see the README or README.html file included in the JUNOScript Perl distribution.

Request and Load Configuration Data

The get_config.pl script described in "Module and Sample Scripts" on page 71 illustrates how to request JUNOScript configuration data. The load_configuration.pl script illustrates how to load configuration data, and includes logic for handling and recovering from errors. They are appropriate bases for custom scripts that perform these functions.

Mapping of Perl Queries to JUNOScript Tags

The sample scripts described in “Module and Sample Scripts” on page 71 invoke only a small number of the predefined JUNOScript Perl queries that client applications can use. Table 6 maps all the Perl queries available in the current version of JUNOScript to the corresponding JUNOScript response tag and JUNOS CLI command. Each query has the same name as the corresponding JUNOScript request tag (to derive the request tag name, replace each underscore with a hyphen and enclose the string in angle brackets).

For more information about JUNOScript request and response tags, see the *JUNOScript API Reference*.

Table 6: Mapping of Perl Queries to Response Tags and CLI Commands

Perl Query	JUNOScript Response Tag	CLI Command
clear_helper_statistics_information	NONE	clear helper statistics
clear_ipv6_nd_information	<ipv6-modify-nd>	clear ipv6 neighbors
file_compare	NONE	file compare
file_copy	NONE	file copy
file_delete	NONE	file delete
file_list	NONE	file list
file_rename	NONE	file rename
file_show	NONE	file show
get_accounting_profile_information	<accounting-profile-information>	show accounting
get_accounting_record_information	<accounting-record-information>	show accounting records
get_alarm_information	<alarm-information>	show chassis alarms
get_bgp_group_information	<bgp-group-information>	show bgp group
get_bgp_neighbor_information	<bgp-information>	show bgp neighbor
get_bgp_summary_information	<bgp-information>	show bgp summary
get_chassis_inventory	<chassis-inventory>	show chassis hardware
get_cos_classifier_information	<cos-classifier-information>	show class-of-service classifier
get_cos_classifier_table_information	<cos-classifier-table-information>	show class-of-service forwarding-table classifier
get_cos_classifier_table_map_information	<cos-classifier-table-map-information>	show class-of-service forwarding-table classifier mapping
get_cos_code_point_map_information	<cos-code-point-map-information>	show class-of-service code-point-aliases
get_cos_drop_profile_information	<cos-drop-profile-information>	show class-of-service drop-profile
get_cos_forwarding_class_information	<cos-forwarding-class-information>	show class-of-service forwarding-class
get_cos_information	<cos-information>	show class-of-service
get_cos_interface_map_information	<cos-interface-information>	show class-of-service interface
get_cos_red_information	<cos-red-information>	show class-of-service forwarding-table drop-profile
get_cos_rewrite_information	<cos-rewrite-information>	show class-of-service rewrite-rule
get_cos_rewrite_table_information	<cos-rewrite-table-information>	show class-of-service forwarding-table rewrite-rule
get_cos_rewrite_table_map_information	<cos-rewrite-table-map-information>	show class-of-service forwarding-table rewrite-rule mapping
get_cos_scheduler_map_information	<cos-scheduler-map-information>	show class-of-service scheduler-map
get_cos_scheduler_map_table_information	<cos-scheduler-map-table-information>	show class-of-service forwarding-table scheduler-map
get_cos_table_information	<cos-table-information>	show class-of-service forwarding-table
get_destination_class_statistics	<destination-class-statistics>	show interfaces destination-class
get_environment_information	<environment-information>	show chassis environment

Perl Query	JUNOScript Response Tag	CLI Command
get_fabric_queue_information	<fabric-queue-information>	show class-of-service fabric queue
get_feb_information	<scb-information>	show chassis feb
get_firewall_information	<firewall-information>	show firewall
get_firewall_log_information	<firewall-log-information>	show firewall log
get_firmware_information	<firmware-information>	show chassis firmware
get_forwarding_table_information	<forwarding-table-information>	show route forwarding-table
get_fpc_information	<fpc-information>	show chassis fpc
get_helper_statistics_information	<helper-statistics-information>	show helper statistics
get_ike_security_associations_information	<ike-security-associations-information>	show ike security-associations
get_instance_information	<instance-information>	show route instance detail
get_instance_summary_information	<instance-information>	show route instance
get_interface_filter_information	<interface-filter-information>	show interfaces filters
get_interface_information	<interface-information>	show interfaces
get_interface_policer_information	<interface-policer-information>	show interfaces policers
get_interface_queue_information	<interface-queue-information>	show interfaces queue
get_ipv6_nd_information	<ipv6-nd-information>	show ipv6 neighbors
get_ipv6_ra_information	<ipv6-ra-information>	show ipv6 router-advertisement
get_isis_adjacency_information	<isis-adjacency-information>	show isis adjacency
get_isis_database_information	<isis-database-information>	show isis database
get_isis_hostname_information	<isis-hostname-information>	show isis hostname
get_isis_interface_information	<isis-interface-information>	show isis interface
get_isis_route_information	<isis-route-information>	show isis route
get_isis_spf_information	<isis-spf-information>	show isis spf
get_isis_statistics_information	<isis-statistics-information>	show isis statistics
get_l2ckt_connection_information	<l2ckt-connection-information>	show l2circuit connections
get_l2vpn_connection_information	<l2vpn-connection-information>	show l2vpn connections
get_ldp_database_information	<ldp-database-information>	show ldp database
get_ldp_interface_information	<ldp-interface-information>	show ldp interface
get_ldp_neighbor_information	<ldp-neighbor-information>	show ldp neighbor
get_ldp_path_information	<ldp-path-information>	show ldp path
get_ldp_route_information	<ldp-route-information>	show ldp route
get_ldp_session_information	<ldp-session-information>	show ldp session
get_ldp_statistics_information	<ldp-statistics-information>	show ldp statistics
get_ldp_traffic_statistics_information	<ldp-traffic-statistics-information>	show ldp traffic-statistics
get_ldp_information	<ldp-information>	show ldp
get_lm_information	<lm-information>	show link-management
get_lm_peer_information	<lm-peer-information>	show link-management peer
get_lm_routing_information	<lm-routing-information>	show link-management routing
get_lm_routing_peer_information	<lm-peer-information>	show link-management routing peer
get_lm_routing_te_link_information	<lm-te-link-information>	show link-management routing te-link
get_lm_te_link_information	<lm-te-link-information>	show link-management te-link
get_mpls_admin_group_information	<mpls-admin-group-information>	show mpls admin-groups
get_mpls_cspf_information	<mpls-cspf-information>	show mpls cspf

Perl Query	JUNOScript Response Tag	CLI Command
get_mpls_interface_information	<mpls-interface-information>	show mpls interface
get_mpls_lsp_information	<mpls-lsp-information>	show mpls lsp
get_mpls_path_information	<mpls-path-information>	show mpls path
get_ospf_database_information	<ospf-database-information>	show ospf database
get_ospf_interface_information	<ospf-interface-information>	show ospf interface
get_ospf_io_statistics_information	<ospf-io-statistics-information>	show ospf io-statistics
get_ospf_log_information	<ospf-log-information>	show ospf log
get_ospf_neighbor_information	<ospf-neighbor-information>	show ospf neighbor
get_ospf_route_information	<ospf-route-information>	show ospf route
get_ospf_statistics_information	<ospf-statistics-information>	show ospf statistics
get_pic_information	<fpc-information>	show chassis fpc pic-status
get_rmon_alarm_information	<rmon-alarm-information>	show snmp rmon alarms
get_rmon_event_information	<rmon-event-information>	show snmp rmon events
get_rmon_information	<rmon-information>	show snmp rmon
get_route_engine_information	<route-engine-information>	show chassis routing-engine
get_route_information	<route-information>	show route
get_route_summary_information	<route-summary-information>	show route summary
get_rsvp_interface_information	<rsvp-interface-information>	show rsvp interface
get_rsvp_neighbor_information	<rsvp-neighbor-information>	show rsvp neighbor
get_rsvp_session_information	<rsvp-session-information>	show rsvp session
get_rsvp_statistics_information	<rsvp-statistics-information>	show rsvp statistics
get_rsvp_version_information	<rsvp-version-information>	show rsvp version
get_rtexport_instance_information	<rtexport-instance-information>	show route export instance
get_rtexport_table_information	<rtexport-table-information>	show route export
get_rtexport_target_information	<rtexport-target-information>	show route export vrf-target
get_scb_information	<scb-information>	show chassis scb
get_security_associations_information	<security-associations-information>	show ipsec security-associations
get_sfm_information	<scb-information>	show chassis sfm
get_snmp_information	<snmp-statistics>	show snmp statistics
get_source_class_statistics	<source-class-statistics>	show interfaces source-class
get_spmb_information	<spmb-information>	show chassis spmb
get_spmb_sib_information	<spmb-sib-information>	show chassis spmb sibs
get_ssb_information	<scb-information>	show chassis ssb
get_syslog_tag_information	<syslog-tag-information>	help syslog
get_ted_database_information	<ted-database-information>	show ted database
get_ted_link_information	<ted-link-information>	show ted link
get_ted_protocol_information	<ted-protocol-information>	show ted protocol
request_halt	NONE	request system halt
request_package_add	NONE	request system software add
request_package_delete	NONE	request system software delete
request_package_validate	NONE	request system software validate
request_reboot	NONE	request system reboot

Chapter 6
Write a C Client Application

The following example illustrates how a client application written in C can use the secure shell (ssh) or telnet protocol to establish a JUNOScript connection and session. In the line that begins with the string execlp, the client application invokes the ssh command. (Substitute the telnet command if appropriate.) The *router* argument to the execlp routine specifies the hostname or IP address of the JUNOScript server. The junoscript argument is the command that converts the connection to a JUNOScript session.

For more information about JUNOScript sessions, see “Start, Control, and End a JUNOScript Session” on page 14.

```
int ipipes[ 2 ], opipes[ 2 ];
pid_t pid;
int rc;
char buf[ BUFSIZ ];

if (pipe(ipipes) <0 || pipe(opipes) <0)
    err(1, "pipe failed");

pid = fork();
if (pid <0)
    err(1, "fork failed");

if (pid == 0) {
    dup2(opipes[ 0 ], STDIN_FILENO);
    dup2(ipipes[ 1 ], STDOUT_FILENO);
    dup2(ipipes[ 1 ], STDERR_FILENO);
    close(ipipes[ 0 ]); /* close read end of pipe */
    close(ipipes[ 1 ]); /* close write end of pipe */
    close(opipes[ 0 ]); /* close read end of pipe */
    close(opipes[ 1 ]); /* close write end of pipe */

    execlp("ssh", "ssh", "-x", router, "junoscript", NULL);
    err (1, "unable to execute: ssh %s junoscript," router);
}

close(ipipes[ 1 ]); /* close write end of pipe */
close(opipes[ 0 ]); /* close read end of pipe */

if (write(opipes[ 1 ], initial_handshake, strlen(initial_handshake)) <0 )
    err(1, "writing initial handshake failed");

rc=read(ipipes[ 0 ], buf, sizeof(buf));
if (rc <0)
    err(1, "read initial handshake failed");
```

Part 4
Index

■ Index on page 81

Index

Index